Package: glmnet (via r-universe)

October 28, 2026

Type Package

Title Lasso and Elastic-Net Regularized Generalized Linear Models

Version 5.2

Date 2026-10-26

Depends R (>= 3.6.0), Matrix (>= 1.0-6)

Imports methods, utils, foreach, shape, survival, Rcpp

Suggests knitr, lars, nnet, testthat, xfun, rmarkdown

SystemRequirements C++17

Description Extremely efficient procedures for fitting the entire lasso or elastic-net regularization path for linear regression, logistic and multinomial regression models, Poisson regression, Cox model, multiple-response Gaussian, and the grouped multinomial regression; see <doi:10.18637/jss.v033.i01> and <doi:10.18637/jss.v039.i05>. There are two new and important additions. The family argument can be a GLM family object, which opens the door to any programmed family (<doi:10.18637/jss.v106.i01>). This comes with a modest computational cost, so when the built-in families suffice, they should be used instead. The other novelty is the relax option, which refits each of the active sets in the path unpenalized. The algorithm uses cyclical coordinate descent in a path-wise fashion, as described in the papers cited.

License GPL-2

VignetteBuilder knitr

Encoding UTF-8

URL https://glmnet.stanford.edu

LinkingTo RcppEigen, Rcpp

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Jerome Friedman [aut], Trevor Hastie [aut, cre], Rob Tibshirani [aut], Balasubramanian Narasimhan [aut], Kenneth Tay [aut], Noah Simon [aut], Junyang Qian [ctb], James Yang [aut], Jonathan Taylor [aut]

Maintainer Trevor Hastie <hastie@stanford.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-28 14:10:44 UTC

RemoteUrl https://github.com/cran/glmnet

RemoteRef HEAD

RemoteSha 37f9d27dc16194893c946582e94feb51fdcd6670

Contents

glmnet-package . . . 3
assess.glmnet . . . 4
beta_CVX . . . 7
bigGlm . . . 7
BinomialExample . . . 8
Cindex . . . 9
coef.glmnet . . . 10
CoxExample . . . 12
coxgrad . . . 13
coxnet.deviance . . . 14
cv.glmnet . . . 16
dev_function . . . 21
deviance.glmnet . . . 21
elnet.fit . . . 23
fid . . . 25
get_eta . . . 26
get_start . . . 26
glmnet . . . 27
glmnet.control . . . 36
glmnet.fit . . . 39
glmnet.measures . . . 42
glmnet.path . . . 43
makeX . . . 45
MultiGaussianExample . . . 47
MultinomialExample . . . 48
mycoxph . . . 48
mycoxpred . . . 49
na.replace . . . 49
obj_function . . . 50
pen_function . . . 51
plot.cv.glmnet . . . 52
plot.glmnet . . . 53
PoissonExample . . . 55

predict.cv.glmnet 56
predict.glmnetfit 57
print.cv.glmnet 59
print.glmnet 60
QuickStartExample 61
response.coxnet 61
rmult 62
SparseExample 62
stratifySurv 63
survfit.coxnet 63
survfit.cv.glmnet 65
weighted_mean_sd 66

Index **67**

`glmnet-package` *Elastic net model paths for some generalized linear models*

Description

This package fits lasso and elastic-net model paths for regression, logistic and multinomial regression using coordinate descent. The algorithm is extremely fast, and exploits sparsity in the input x matrix where it exists. A variety of predictions can be made from the fitted models.

Details

Package:	glmnet
Type:	Package
Version:	1.0
Date:	2008-05-14
License:	What license is it under?

Very simple to use. Accepts `x,y` data for regression models, and produces the regularization path over a grid of values for the tuning parameter `lambda`. Only 5 functions: `glmnet`
`predict.glmnet`
`plot.glmnet`
`print.glmnet`
`coef.glmnet`

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.
Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.
Tibshirani,Robert, Bien, J., Friedman, J., Hastie, T.,Simon, N.,Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems, JRSSB, Vol. 74(2), 245-266*, https://arxiv.org/abs/1011.2234.
Hastie, T., Tibshirani, Robert and Tibshirani, Ryan (2020) *Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons, Statist. Sc. Vol. 35(4), 579-592*, https://arxiv.org/abs/1707.08692.
Glmnet webpage with four vignettes: https://glmnet.stanford.edu.

See Also

Useful links:

- https://glmnet.stanford.edu

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
g2 = sample(1:2, 100, replace = TRUE)
g4 = sample(1:4, 100, replace = TRUE)
fit1 = glmnet(x, y)
predict(fit1, newx = x[1:5, ], s = c(0.01, 0.005))
predict(fit1, type = "coef")
plot(fit1, xvar = "lambda")
fit2 = glmnet(x, g2, family = "binomial")
predict(fit2, type = "response", newx = x[2:5, ])
predict(fit2, type = "nonzero")
fit3 = glmnet(x, g4, family = "multinomial")
predict(fit3, newx = x[1:3, ], type = "response", s = 0.01)
```

`assess.glmnet` *assess performance of a 'glmnet' object using test data.*

Description

Given a test set, produce summary performance measures for the glmnet model(s)

Usage

```
assess.glmnet(
  object,
  newx = NULL,
  newy,
  weights = NULL,
 family = c("gaussian", "binomial", "poisson", "multinomial", "cox", "mgaussian"),
  ...
)

confusion.glmnet(
  object,
  newx = NULL,
  newy,
  family = c("binomial", "multinomial"),
  ...
)

roc.glmnet(object, newx = NULL, newy, ...)
```

Arguments

object	Fitted "glmnet" or "cv.glmnet", "relaxed" or "cv.relaxed" object, OR a matrix of predictions (for roc.glmnet or assess.glmnet). For roc.glmnet the model must be a 'binomial', and for confusion.glmnet must be either 'binomial' or 'multinomial'
newx	If predictions are to made, these are the 'x' values. Required for confusion.glmnet
newy	required argument for all functions; the new response values
weights	For observation weights for the test observations
family	The family of the model, in case predictions are passed in as 'object'
...	additional arguments to predict.glmnet when "object" is a "glmnet" fit, and predictions must be made to produce the statistics.

Details

assess.glmnet produces all the different performance measures provided by cv.glmnet for each of the families. A single vector, or a matrix of predictions can be provided, or fitted model objects or CV objects. In the case when the predictions are still to be made, the ... arguments allow, for example, 'offsets' and other prediction parameters such as values for 'gamma' for 'relaxed' fits. roc.glmnet produces for a single vector a two column matrix with columns TPR and FPR (true positive rate and false positive rate). This object can be plotted to produce an ROC curve. If more than one predictions are called for, then a list of such matrices is produced. confusion.glmnet produces a confusion matrix tabulating the classification results. Again, a single table or a list, with a print method.

Value

`assess.glmnet` produces a list of vectors of measures. `roc.glmnet` a list of 'roc' two-column matrices, and `confusion.glmnet` a list of tables. If a single prediction is provided, or predictions are made from a CV object, the latter two drop the list status and produce a single matrix or table.

Author(s)

Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`cv.glmnet`, `glmnet.measures` and `vignette("relax",package="glmnet")`

Examples

```
data(QuickStartExample)
x <- QuickStartExample$x; y <- QuickStartExample$y
set.seed(11)
train = sample(seq(length(y)),70,replace=FALSE)
fit1 = glmnet(x[train,], y[train])
assess.glmnet(fit1, newx = x[-train,], newy = y[-train])
preds = predict(fit1, newx = x[-train, ], s = c(1, 0.25))
assess.glmnet(preds, newy = y[-train], family = "gaussian")
fit1c = cv.glmnet(x, y, keep = TRUE)
fit1a = assess.glmnet(fit1c$fit.preval, newy=y,family="gaussian")
plot(fit1c$lambda, log="x",fit1a$mae,xlab="Log Lambda",ylab="Mean Absolute Error")
abline(v=fit1c$lambda.min, lty=2, col="red")
data(BinomialExample)
x <- BinomialExample$x; y <- BinomialExample$y
fit2 = glmnet(x[train,], y[train], family = "binomial")
assess.glmnet(fit2,newx = x[-train,], newy=y[-train], s=0.1)
plot(roc.glmnet(fit2, newx = x[-train,], newy=y[-train])[[10]])
fit2c = cv.glmnet(x, y, family = "binomial", keep=TRUE)
idmin = match(fit2c$lambda.min, fit2c$lambda)
plot(roc.glmnet(fit2c$fit.preval, newy = y)[[idmin]])
data(MultinomialExample)
x <- MultinomialExample$x; y <- MultinomialExample$y
set.seed(103)
train = sample(seq(length(y)),100,replace=FALSE)
fit3 = glmnet(x[train,], y[train], family = "multinomial")
confusion.glmnet(fit3, newx = x[-train, ], newy = y[-train], s = 0.01)
fit3c = cv.glmnet(x, y, family = "multinomial", type.measure="class", keep=TRUE)
idmin = match(fit3c$lambda.min, fit3c$lambda)
confusion.glmnet(fit3c$fit.preval, newy = y, family="multinomial")[[idmin]]
```

beta_CVX *Simulated data for the glmnet vignette*

Description

Simple simulated data, used to demonstrate the features of glmnet

Format

Data objects used to demonstrate features in the glmnet vignette

Details

These datasets are artificial, and are used to test out some of the features of glmnet.

Examples

```
data(QuickStartExample)
x <- QuickStartExample$x; y <- QuickStartExample$y
glmnet(x, y)
```

bigGlm *fit a glm with all the options in* glmnet

Description

Fit a generalized linear model as in glmnet but unpenalized. This allows all the features of glmnet such as sparse x, bounds on coefficients, offsets, and so on.

Usage

```
bigGlm(x, ..., path = FALSE)
```

Arguments

x	input matrix
...	Most other arguments to glmnet that make sense
path	Since glmnet does not do stepsize optimization, the Newton algorithm can get stuck and not converge, especially with unpenalized fits. With path=TRUE, the fit computed with pathwise lasso regularization. The current implementation does this twice: the first time to get the lambda sequence, and the second time with a zero attached to the end). Default is path=FALSE.

Details

This is essentially the same as fitting a "glmnet" model with a single value `lambda=0`, but it avoids some edge cases. CAVEAT: If the user tries a problem with N smaller than or close to p for some models, it is likely to fail (and maybe not gracefully!) If so, use the `path=TRUE` argument.

Value

It returns an object of class "bigGlm" that inherits from class "glmnet". That means it can be predicted from, coefficients extracted via `coef`. It has its own print method.

Author(s)

Trevor Hastie
Maintainer: Trevor Hastie <hastie@stanford.edu>

See Also

`print`, `predict`, and `coef` methods.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = bigGlm(x, y)
print(fit1)

fit2=bigGlm(x,y>0,family="binomial")
print(fit2)
fit2p=bigGlm(x,y>0,family="binomial",path=TRUE)
print(fit2p)
```

BinomialExample *Synthetic dataset with binary response*

Description

Randomly generated data for binomial regression example.

Usage

```
data(BinomialExample)
```

Format

List containing the following elements:

x 100 by 30 matrix of numeric values.

y Numeric vector of length 100 containing 44 zeros and 56 ones.

Cindex *compute C index for a Cox model*

Description

Computes Harrel's C index for predictions from a "coxnet" object.

Usage

```
Cindex(pred, y, weights = rep(1, nrow(y)))
```

Arguments

pred	Predictions from a "coxnet" object
y	a survival response object - a matrix with two columns "time" and "status"; see documentation for "glmnet"
weights	optional observation weights

Details

Computes the concordance index, taking into account censoring.

Author(s)

Trevor Hastie hastie@stanford.edu

References

Harrel Jr, F. E. and Lee, K. L. and Mark, D. B. (1996) *Tutorial in biostatistics: multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing error*, Statistics in Medicine, 15, pages 361–387.

See Also

cv.glmnet

Examples

```
set.seed(10101)
N = 1000
p = 30
nzc = p/3
x = matrix(rnorm(N * p), N, p)
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta/3
hx = exp(fx)
ty = rexp(N, hx)
tcens = rbinom(n = N, prob = 0.3, size = 1)  # censoring indicator
```

```
y = cbind(time = ty, status = 1 - tcens)  # y=Surv(ty,1-tcens) with library(survival)
fit = glmnet(x, y, family = "cox")
pred = predict(fit, newx = x)
apply(pred, 2, Cindex, y=y)
cv.glmnet(x, y, family = "cox", type.measure = "C")
```

`coef.glmnet` *Extract coefficients from a glmnet object*

Description

Similar to other predict methods, this functions predicts fitted values, logits, coefficients and more from a fitted `"glmnet"` object.

Usage

```
## S3 method for class 'glmnet'
coef(object, s = NULL, exact = FALSE, ...)

## S3 method for class 'glmnet'
predict(
  object,
  newx,
  s = NULL,
  type = c("link", "response", "coefficients", "nonzero", "class"),
  exact = FALSE,
  newoffset,
  ...
)

## S3 method for class 'relaxed'
predict(
  object,
  newx,
  s = NULL,
  gamma = 1,
  type = c("link", "response", "coefficients", "nonzero", "class"),
  exact = FALSE,
  newoffset,
  ...
)
```

Arguments

`object`	Fitted `"glmnet"` model object or a `"relaxed"` model (which inherits from class "glmnet").

s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model.
exact	This argument is relevant only when predictions are made at values of s (lambda) *different* from those used in the fitting of the original model. Not available for "relaxed" objects. If exact=FALSE (default), then the predict function uses linear interpolation to make predictions for values of s (lambda) that do not coincide with those used in the fitting algorithm. While this is often a good approximation, it can sometimes be a bit coarse. With exact=TRUE, these different values of s are merged (and sorted) with object$lambda, and the model is refit before predictions are made. In this case, it is required to supply the original data x= and y= as additional named arguments to predict() or coef(). The workhorse predict.glmnet() needs to update the model, and so needs the data used to create it. The same is true of weights, offset, penalty.factor, lower.limits, upper.limits if these were used in the original call. Failure to do so will result in an error.
...	This is the mechanism for passing arguments like x= when exact=TRUE; seeexact argument.
newx	Matrix of new values for x at which predictions are to be made. Must be a matrix; can be sparse as in Matrix package. This argument is not used for type=c("coefficients","nonzero")
type	Type of prediction required. Type "link" gives the linear predictors for "binomial", "multinomial", "poisson" or "cox" models; for "gaussian" models it gives the fitted values. Type "response" gives the fitted probabilities for "binomial" or "multinomial", fitted mean for "poisson" and the fitted relative-risk for "cox"; for "gaussian" type "response" is equivalent to type "link". Type "coefficients" computes the coefficients at the requested values for s. Note that for "binomial" models, results are returned only for the class corresponding to the second level of the factor response. Type "class" applies only to "binomial" or "multinomial" models, and produces the class label corresponding to the maximum probability. Type "nonzero" returns a list of the indices of the nonzero coefficients for each value of s.
newoffset	If an offset is used in the fit, then one must be supplied for making predictions (except for type="coefficients" or type="nonzero")
gamma	Value(s) of gamma at which predictions are required, for "relaxed" objects. If more than one value is given, a list of predictions is returned, one per gamma, named by the rounded gamma values.

Details

The shape of the objects returned are different for "multinomial" objects. This function actually calls NextMethod(), and the appropriate predict method is invoked for each of the three model types. coef(...) is equivalent to predict(type="coefficients",...)

Value

The object returned depends on type.

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.
Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.
Glmnet webpage with four vignettes, https://glmnet.stanford.edu.

See Also

`glmnet`, and `print`, and `coef` methods, and `cv.glmnet`.

Examples

```
x=matrix(rnorm(100*20),100,20)
y=rnorm(100)
g2=sample(1:2,100,replace=TRUE)
g4=sample(1:4,100,replace=TRUE)
fit1=glmnet(x,y)
predict(fit1,newx=x[1:5,],s=c(0.01,0.005))
predict(fit1,type="coef")
fit2=glmnet(x,g2,family="binomial")
predict(fit2,type="response",newx=x[2:5,])
predict(fit2,type="nonzero")
fit3=glmnet(x,g4,family="multinomial")
predict(fit3,newx=x[1:3,],type="response",s=0.01)
```

CoxExample *Synthetic dataset with right-censored survival response*

Description

Randomly generated data for Cox regression example.

Usage

```
data(CoxExample)
```

Format

List containing the following elements:

x 1,000 by 30 matrix of numeric values.

y 1,000 by 2 matrix with column names "time" and "status". The first column consists of positive numbers representing time to event, while the second column represents the status indicator (0=right-censored, 1=observed).

`coxgrad` *Compute gradient for Cox model*

Description

Compute the gradient of the log partial likelihood at a particular fit for Cox model.

Usage

```
coxgrad(
  eta,
  y,
  w,
  std.weights = TRUE,
  diag.hessian = FALSE,
  cox.ties = c("breslow", "efron")
)
```

Arguments

`eta`	Fit vector (usually from glmnet at a particular lambda).
`y`	Survival response variable, must be a `Surv` or `stratifySurv` object.
`w`	Observation weights (default is all equal to 1).
`std.weights`	If TRUE (default), observation weights are standardized to sum to 1.
`diag.hessian`	If `TRUE`, compute the diagonal of the Hessian of the deviance (minus twice the log partial likelihood) as well. Default is `FALSE`.
`cox.ties`	Character; the method for handling ties. One of `"breslow"` (the current default) or `"efron"`. The default will change to `"efron"` in glmnet 5.1 to match `survival::coxph`.

Details

Compute a gradient vector at the fitted vector for the log partial likelihood. This is like a residual vector, and useful for manual screening of predictors for `glmnet` in applications where `p` is very large (as in GWAS).

Uses the C++ coxdev library for computation, supporting both Breslow and Efron methods for ties, as well as stratified and (start, stop] data.

Value

A single gradient vector the same length as `eta`. If `diag.hessian=TRUE`, the diagonal of the Hessian of the deviance with respect to `eta` is included as an attribute "diag_hessian". It is on the deviance scale while the gradient is on the log partial likelihood scale, so it equals -2 times the derivative of the gradient.

See Also

`coxnet.deviance`

Examples

```
set.seed(1)
eta <- rnorm(10)
time <- runif(10, min = 1, max = 10)
d <- ifelse(rnorm(10) > 0, 1, 0)
y <- survival::Surv(time, d)
coxgrad(eta, y)

# return diagonal of Hessian as well
coxgrad(eta, y, diag.hessian = TRUE)

# example with (start, stop] data
y2 <- survival::Surv(time, time + runif(10), d)
coxgrad(eta, y2)

# example with strata
y2 <- stratifySurv(y, rep(1:2, length.out = 10))
coxgrad(eta, y2)
```

---

coxnet.deviance *Compute deviance for Cox model*

Description

Compute the deviance (-2 log partial likelihood) for Cox model.

Usage

```
coxnet.deviance(
  pred = NULL,
  y,
  x = NULL,
  offset = NULL,
  weights = NULL,
  std.weights = TRUE,
  beta = NULL,
  cox.ties = c("breslow", "efron")
)
```

Arguments

pred	Fit vector or matrix (usually from glmnet at a particular lambda or a sequence of lambdas).
y	Survival response variable, must be a Surv or stratifySurv object.
x	Optional x matrix, to be supplied if pred = NULL.
offset	Optional offset vector.
weights	Observation weights (default is all equal to 1).
std.weights	If TRUE (default), observation weights are standardized to sum to 1.
beta	Optional coefficient vector/matrix, to be supplied if pred = NULL.
cox.ties	Character; the method for handling ties. One of "breslow" (the current default) or "efron". The default will change to "efron" in glmnet 5.1 to match survival::coxph.

Details

Computes the deviance for a single set of predictions, or for a matrix of predictions. The user can either supply the predictions directly through the pred option, or by supplying the x matrix and beta coefficients.

The function first checks if pred is passed: if so, it is used as the predictions. If pred is not passed but x and beta are passed, then these values are used to compute the predictions. If neither x nor beta are passed, then the predictions are all taken to be 0.

Uses the C++ coxdev library for computation, supporting both Breslow and Efron methods for ties, as well as stratified and (start, stop] data.

Value

A vector of deviances, one for each column of predictions.

See Also

coxgrad

Examples

```
set.seed(1)
eta <- rnorm(10)
time <- runif(10, min = 1, max = 10)
d <- ifelse(rnorm(10) > 0, 1, 0)
y <- survival::Surv(time, d)
coxnet.deviance(pred = eta, y = y)

# if pred not provided, it is set to zero vector
coxnet.deviance(y = y)

# example with x and beta
x <- matrix(rnorm(10 * 3), nrow = 10)
beta <- matrix(1:3, ncol = 1)
```

```
coxnet.deviance(y = y, x = x, beta = beta)

# example with (start, stop] data
y2 <- survival::Surv(time, time + runif(10), d)
coxnet.deviance(pred = eta, y = y2)

# example with strata
y2 <- stratifySurv(y, rep(1:2, length.out = 10))
coxnet.deviance(pred = eta, y = y2)
```

---

cv.glmnet *Cross-validation for glmnet*

Description

Does k-fold cross-validation for glmnet, produces a plot, and returns a value for `lambda` (and `gamma` if `relax=TRUE`)

Usage

```
cv.glmnet(
  x,
  y,
  weights = NULL,
  offset = NULL,
  lambda = NULL,
  type.measure = c("default", "mse", "deviance", "class", "auc", "mae", "C"),
  nfolds = 10,
  foldid = NULL,
  alignment = c("lambda", "fraction"),
  grouped = TRUE,
  keep = FALSE,
  parallel = FALSE,
  gamma = c(0, 0.25, 0.5, 0.75, 1),
  relax = FALSE,
  trace.it = 0,
  control = list(),
  ...
)
```

Arguments

`x`	x matrix as in `glmnet`.
`y`	response `y` as in `glmnet`.
`weights`	Observation weights; defaults to 1 per observation
`offset`	Offset vector (matrix) as in `glmnet`

`lambda` Optional user-supplied lambda sequence; default is `NULL`, and `glmnet` chooses its own sequence. Note that this is done for the full model (master sequence), and separately for each fold. The fits are then alligned using the master sequence (see the `allignment` argument for additional details). Adapting `lambda` for each fold leads to better convergence. When `lambda` is supplied, the same sequence is used everywhere, but in some GLMs can lead to convergence issues.

`type.measure` loss to use for cross-validation. Currently five options, not all available for all models. The default is `type.measure="deviance"`, which uses squared-error for gaussian models (a.k.a `type.measure="mse"` there), deviance for logistic and poisson regression, and partial-likelihood for the Cox model. `type.measure="class"` applies to binomial and multinomial logistic regression only, and gives misclassification error. `type.measure="auc"` is for two-class logistic regression only, and gives area under the ROC curve. `type.measure="mse"` or `type.measure="mae"` (mean absolute error) can be used by all models except the `"cox"`; they measure the deviation from the fitted mean to the response. For binomial model and binary data, `type.measure="mse"` amounts to the "Brier" score. `type.measure="C"` is Harrel's concordance measure, only available for `cox` models.

`nfolds` number of folds - default is 10. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds=3`

`foldid` an optional vector of values between 1 and `nfolds` identifying what fold each observation is in. If supplied, `nfolds` can be missing.

`alignment` This is an experimental argument, designed to fix the problems users were having with CV, with possible values `"lambda"` (the default) else `"fraction"`. With `"lambda"` the `lambda` values from the master fit (on all the data) are used to line up the predictions from each of the folds. In some cases this can give strange values, since the effective `lambda` values in each fold could be quite different. With `"fraction"` we line up the predictions in each fold according to the fraction of progress along the regularization. If in the call a `lambda` argument is also provided, `alignment="fraction"` is ignored (with a warning).

`grouped` This is an experimental argument, with default `TRUE`, and can be ignored by most users. For all models except the `"cox"`, this refers to computing `nfolds` separate statistics, and then using their mean and estimated standard error to describe the CV curve. If `grouped=FALSE`, an error matrix is built up at the observation level from the predictions from the `nfolds` fits, and then summarized (does not apply to `type.measure="auc"`). For the `"cox"` family, `grouped=TRUE` obtains the CV partial likelihood for the Kth fold by *subtraction*; by subtracting the log partial likelihood evaluated on the full dataset from that evaluated on the on the (K-1)/K dataset. This makes more efficient use of risk sets. With `grouped=FALSE` the log partial likelihood is computed only on the Kth fold

`keep` If `keep=TRUE`, a *prevalidated* array is returned containing fitted values for each observation and each value of `lambda`. This means these fits are computed with this observation and the rest of its fold omitted. The `foldid` vector is also returned. Default is keep=FALSE. If `relax=TRUE`, then a list of such arrays is returned, one for each value of 'gamma'. Note: if the value 'gamma=1' is omitted, this case is included in the list since it corresponds to the original 'glmnet' fit.

parallel	If TRUE, use parallel foreach to fit each fold. Must register parallel before hand, such as doMC or others. See the example below.
gamma	The values of the parameter for mixing the relaxed fit with the regularized fit, between 0 and 1; default is gamma = c(0, 0.25, 0.5,0.75, 1)
relax	If TRUE, then CV is done with respect to the mixing parameter gamma as well as lambda. Default is relax=FALSE
trace.it	If trace.it=1, then progress bars are displayed; useful for big models that take a long time to fit. Limited tracing if parallel=TRUE
control	A named list of algorithm control parameters, providing per-call overrides of session defaults set by glmnet.control(). See glmnet for details.
...	Other arguments that can be passed to glmnet, for example alpha, nlambda, etc. See glmnet for details.

Details

The function runs glmnet nfolds+1 times; the first to get the lambda sequence, and then the remainder to compute the fit with each of the folds omitted. The error is accumulated, and the average error and standard deviation over the folds is computed. Note that cv.glmnet does NOT search for values for alpha. A specific value should be supplied, else alpha=1 is assumed by default. If users would like to cross-validate alpha as well, they should call cv.glmnet with a pre-computed vector foldid, and then use this same fold vector in separate calls to cv.glmnet with different values of alpha. Note also that the results of cv.glmnet are random, since the folds are selected at random. Users can reduce this randomness by running cv.glmnet many times, and averaging the error curves.

If relax=TRUE then the values of gamma are used to mix the fits. If η is the fit for lasso/elastic net, and η_R is the relaxed fit (with unpenalized coefficients), then a relaxed fit mixed by γ is

$$\eta(\gamma) = (1-\gamma)\eta_R + \gamma\eta.$$

There is practically no extra cost for having a lot of values for gamma. However, 5 seems sufficient for most purposes. CV then selects both gamma and lambda.

Value

an object of class "cv.glmnet" is returned, which is a list with the ingredients of the cross-validation fit. If the object was created with relax=TRUE then this class has a prefix class of "cv.relaxed".

lambda	the values of lambda used in the fits.
cvm	The mean cross-validated error - a vector of length length(lambda).
cvsd	estimate of standard error of cvm.
cvup	upper curve = cvm+cvsd.
cvlo	lower curve = cvm-cvsd.
nzero	number of non-zero coefficients at each lambda.
name	a text string indicating type of measure (for plotting purposes).
glmnet.fit	a fitted glmnet object for the full data.

lambda.min	value of lambda that gives minimum cvm.
lambda.1se	largest value of lambda such that error is within 1 standard error of the minimum.
fit.preval	if keep=TRUE, this is the array of prevalidated fits. Some entries can be NA, if that and subsequent values of lambda are not reached for that fold
foldid	if keep=TRUE, the fold assignments used
index	a one column matrix with the indices of lambda.min and lambda.1se in the sequence of coefficients, fits etc.
relaxed	if relax=TRUE, this additional item has the CV info for each of the mixed fits. In particular it also selects lambda, gamma pairs corresponding to the 1se rule, as well as the minimum error. It also has a component index, a two-column matrix which contains the lambda and gamma indices corresponding to the "min" and "1se" solutions.

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Noah Simon helped develop the 'coxnet' function.
Jeffrey Wong and B. Narasimhan helped with the parallel option
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.
Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.

See Also

glmnet and plot, predict, and coef methods for "cv.glmnet" and "cv.relaxed" objects.

Examples

```
set.seed(1010)
n = 1000
p = 100
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta
eps = rnorm(n) * 5
y = drop(fx + eps)
px = exp(fx)
px = px/(1 + px)
ly = rbinom(n = length(px), prob = px, size = 1)
set.seed(1011)
```

```
cvob1 = cv.glmnet(x, y)
plot(cvob1)
coef(cvob1)
predict(cvob1, newx = x[1:5, ], s = "lambda.min")
title("Gaussian Family", line = 2.5)
set.seed(1011)
cvob1a = cv.glmnet(x, y, type.measure = "mae")
plot(cvob1a)
title("Gaussian Family", line = 2.5)
set.seed(1011)
par(mfrow = c(2, 2), mar = c(4.5, 4.5, 4, 1))
cvob2 = cv.glmnet(x, ly, family = "binomial")
plot(cvob2)
title("Binomial Family", line = 2.5)
frame()
set.seed(1011)
cvob3 = cv.glmnet(x, ly, family = "binomial", type.measure = "class")
plot(cvob3)
title("Binomial Family", line = 2.5)
## Not run:
cvob1r = cv.glmnet(x, y, relax = TRUE)
plot(cvob1r)
predict(cvob1r, newx = x[, 1:5])
set.seed(1011)
cvob3a = cv.glmnet(x, ly, family = "binomial", type.measure = "auc")
plot(cvob3a)
title("Binomial Family", line = 2.5)
set.seed(1011)
mu = exp(fx/10)
y = rpois(n, mu)
cvob4 = cv.glmnet(x, y, family = "poisson")
plot(cvob4)
title("Poisson Family", line = 2.5)

# Multinomial
n = 500
p = 30
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
beta3 = matrix(rnorm(30), 10, 3)
beta3 = rbind(beta3, matrix(0, p - 10, 3))
f3 = x %*% beta3
p3 = exp(f3)
p3 = p3/apply(p3, 1, sum)
g3 = glmnet:::rmult(p3)
set.seed(10101)
cvfit = cv.glmnet(x, g3, family = "multinomial")
plot(cvfit)
title("Multinomial Family", line = 2.5)
# Cox
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta/3
hx = exp(fx)
```

```
ty = rexp(n, hx)
tcens = rbinom(n = n, prob = 0.3, size = 1)  # censoring indicator
y = cbind(time = ty, status = 1 - tcens)  # y=Surv(ty,1-tcens) with library(survival)
foldid = sample(rep(seq(10), length = n))
fit1_cv = cv.glmnet(x, y, family = "cox", foldid = foldid)
plot(fit1_cv)
title("Cox Family", line = 2.5)
# Parallel
require(doMC)
registerDoMC(cores = 4)
x = matrix(rnorm(1e+05 * 100), 1e+05, 100)
y = rnorm(1e+05)
system.time(cv.glmnet(x, y))
system.time(cv.glmnet(x, y, parallel = TRUE))

## End(Not run)
```

`dev_function` *Elastic net deviance value*

Description

Returns the elastic net deviance value.

Usage

```
dev_function(y, mu, weights, family)
```

Arguments

`y`	Quantitative response variable.
`mu`	Model's predictions for `y`.
`weights`	Observation weights.
`family`	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function.

`deviance.glmnet` *Extract the deviance from a glmnet object*

Description

Compute the deviance sequence from the glmnet object

Usage

```
## S3 method for class 'glmnet'
deviance(object, ...)
```

Arguments

object	fitted glmnet object
...	additional print arguments

Details

A glmnet object has components dev.ratio and nulldev. The former is the fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Null deviance is defined to be 2*(loglike_sat -loglike(Null)); The NULL model refers to the intercept model, except for the Cox, where it is the 0 model. Hence dev.ratio=1-deviance/nulldev, and this deviance method returns (1-dev.ratio)*nulldev.

Value

(1-dev.ratio)*nulldev

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent*

See Also

glmnet, predict, print, and coef methods.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = glmnet(x, y)
deviance(fit1)
```

`elnet.fit` *Solve weighted least squares (WLS) problem for a single lambda value*

Description

Solves the weighted least squares (WLS) problem for a single lambda value. Internal function that users should not call directly.

Usage

```
elnet.fit(
  x,
  y,
  weights,
  lambda,
  alpha = 1,
  intercept = TRUE,
  penalty.factor = rep(1, nvars),
  exclude = c(),
  lower.limits = -Inf,
  upper.limits = Inf,
  warm = NULL,
  from.glmnet.fit = FALSE,
  save.fit = FALSE,
  control = glmnet.control()
)
```

Arguments

`x`	Input matrix, of dimension `nobs` x `nvars`; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes `xm` and `xs`, where `xm(j)` and `xs(j)` are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed that any standardization needed has already been done.
`y`	Quantitative response variable.
`weights`	Observation weights. `elnet.fit` does NOT standardize these weights.
`lambda`	A single value for the `lambda` hyperparameter.
`alpha`	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2\|\|\beta\|\|_2^2 + \alpha\|\|\beta\|\|_1.$$ `alpha=1` is the lasso penalty, and `alpha=0` the ridge penalty.
`intercept`	Should intercept be fitted (default=TRUE) or set to zero (FALSE)?

`penalty.factor` Separate penalty factors can be applied to each coefficient. This is a number that multiplies `lambda` to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude). Note: the penalty factors are internally rescaled to sum to `nvars`.

`exclude` Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor.

`lower.limits` Vector of lower limits for each coefficient; default `-Inf`. Each of these must be non-positive. Can be presented as a single value (which will then be replicated), else a vector of length `nvars`.

`upper.limits` Vector of upper limits for each coefficient; default `Inf`. See `lower.limits`.

`warm` Either a `glmnetfit` object or a list (with names `beta` and `a0` containing coefficients and intercept respectively) which can be used as a warm start. Default is `NULL`, indicating no warm start. For internal use only.

`from.glmnet.fit` Was `elnet.fit()` called from `glmnet.fit()`? Default is FALSE.This has implications for computation of the penalty factors.

`save.fit` Return the warm start object? Default is FALSE.

`control` A fully resolved 17-key control list of the form returned by `glmnet.control()`. Default is `glmnet.control()` – current session state. This function does not resolve or validate the list; keys (`thresh`, `maxit`, `big`, etc.) are read directly. See `?glmnet.path` for the same contract.

Details

WARNING: Users should not call `elnet.fit` directly. Higher-level functions in this package call `elnet.fit` as a subroutine. If a warm start object is provided, some of the other arguments in the function may be overriden.

`elnet.fit` is essentially a wrapper around a C++ subroutine which minimizes

$$1/2 \sum w_i(y_i - X_i^T\beta)^2 + \sum \lambda\gamma_j[(1-\alpha)/2\beta^2 + \alpha|\beta|],$$

over β, where γ_j is the relative penalty factor on the jth variable. If `intercept = TRUE`, then the term in the first sum is $w_i(y_i - \beta_0 - X_i^T\beta)^2$, and we are minimizing over both β_0 and β.

None of the inputs are standardized except for `penalty.factor`, which is standardized so that they sum up to `nvars`.

Value

An object with class "glmnetfit" and "glmnet". The list returned has the same keys as that of a `glmnet` object, except that it might have an additional `warm_fit` key.

`a0` Intercept value.

`beta` A `nvars` x 1 matrix of coefficients, stored in sparse matrix format.

`df` The number of nonzero coefficients.

`dim` Dimension of coefficient matrix.

lambda	Lambda value used.
dev.ratio	The fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.
nulldev	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)). The null model refers to the intercept model.
npasses	Total passes over the data.
jerr	Error flag, for warnings and errors (largely for internal debugging).
offset	Always FALSE, since offsets do not appear in the WLS problem. Included for compability with glmnet output.
call	The call that produced this object.
nobs	Number of observations.
warm_fit	If save.fit=TRUE, output of C++ routine, used for warm starts. For internal use only.

`fid` *Helper function for Cox deviance and gradient*

Description

Helps to find ties in death times of data.

Usage

```
fid(x, index)
```

Arguments

x	Sorted vector of death times.
index	Vector of indices for the death times.

Value

A list with two arguments.

index_first	A vector of indices for the first observation at each death time as they appear in the sorted list.
index_ties	If there are no ties at all, this is NULL. If not, this is a list with length equal to the number of unique times with ties. For each time with ties, index_ties gives the indices of the observations with a death at that time.

Examples

```
# Example with no ties
glmnet:::fid(c(1, 4, 5, 6), 1:5)

# Example with ties
glmnet:::fid(c(1, 1, 1, 2, 3, 3, 4, 4, 4), 1:9)
```

`get_eta` *Helper function to get etas (linear predictions)*

Description

Given x, coefficients and intercept, return linear predictions. Wrapper that works with both regular and sparse x. Only works for single set of coefficients and intercept.

Usage

```
get_eta(x, beta, a0)
```

Arguments

`x`	Input matrix, of dimension `nobs` x `nvars`; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes `xm` and `xs`, where `xm(j)` and `xs(j)` are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed to be standardized.
`beta`	Feature coefficients.
`a0`	Intercept.

`get_start` *Get null deviance, starting mu and lambda max*

Description

Return the null deviance, starting mu and lambda max values for initialization. For internal use only.

Usage

```
get_start(
  x,
  y,
  weights,
  family,
  intercept,
  is.offset,
```

```
  offset,
  exclude,
  vp,
  alpha
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes xm and xs, where xm(j) and xs(j) are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed to be standardized.
y	Quantitative response variable.
weights	Observation weights.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. (See family for details on family functions.)
intercept	Does the model we are fitting have an intercept term or not?
is.offset	Is the model being fit with an offset or not?
offset	Offset for the model. If is.offset=FALSE, this should be a zero vector of the same length as y.
exclude	Indices of variables to be excluded from the model.
vp	Separate penalty factors can be applied to each coefficient.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$.

Details

This function is called by glmnet.path for null deviance, starting mu and lambda max values. It is also called by glmnet.fit when used without warmstart, but they only use the null deviance and starting mu values.

When x is not sparse, it is expected to already by centered and scaled. When x is sparse, the function will get its attributes xm and xs for its centering and scaling factors.

Note that whether x is centered & scaled or not, the values of mu and nulldev don't change. However, the value of lambda_max does change, and we need xm and xs to get the correct value.

---

glmnet *fit a GLM with lasso or elasticnet regularization*

Description

Fit a generalized linear model via penalized maximum likelihood. The regularization path is computed for the lasso or elasticnet penalty at a grid of values for the regularization parameter lambda. Can deal with all shapes of data, including very large sparse data matrices. Fits linear, logistic and multinomial, poisson, and Cox regression models.

Usage

```
glmnet(
  x,
  y,
 family = c("gaussian", "binomial", "poisson", "multinomial", "cox", "mgaussian"),
  weights = NULL,
  offset = NULL,
  alpha = 1,
  nlambda = 100,
  lambda.min.ratio = ifelse(nobs < nvars, 0.01, 1e-04),
  lambda = NULL,
  standardize = TRUE,
  intercept = TRUE,
  thresh = 1e-07,
  dfmax = NULL,
  pmax = NULL,
  exclude = NULL,
  penalty.factor = rep(1, nvars),
  lower.limits = -Inf,
  upper.limits = Inf,
  maxit = 1e+05,
  type.gaussian = ifelse(nvars < 500, "covariance", "naive"),
  type.logistic = c("Newton", "modified.Newton"),
  standardize.response = FALSE,
  type.multinomial = c("ungrouped", "grouped"),
  relax = FALSE,
  trace.it = 0,
  cox.ties = c("breslow", "efron"),
  control = list(),
  ...
)

relax.glmnet(fit, x, ..., maxp = n - 3, path = FALSE, check.args = TRUE)
```

Arguments

x	input matrix, of dimension nobs x nvars; each row is an observation vector. Can be in sparse matrix format (inherit from class "sparseMatrix" as in package Matrix). Requirement: nvars >1; in other words, x should have 2 or more columns.
y	response variable. Quantitative for family="gaussian", or family="poisson" (non-negative counts). For family="binomial" should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). For family="multinomial", can be a nc>=2 level factor, or a matrix with nc columns of counts or proportions. For either "binomial" or "multinomial", if y is presented as a vector, it will be coerced into a factor. For family="cox", preferably a Surv object from the survival package: see De-

tails section for more information. For `family="mgaussian"`, `y` is a matrix of quantitative responses.

`family` Either a character string representing one of the built-in families, or else a `glm()` family object. For more information, see Details section below or the documentation for response type (above).

`weights` observation weights. Can be total counts if responses are proportion matrices. Default is 1 for each observation

`offset` A vector of length `nobs` that is included in the linear predictor (a `nobs` x `nc` matrix for the `"multinomial"` family). Useful for the `"poisson"` family (e.g. log of exposure time), or for refining a model by starting at a current fit. Default is `NULL`. If supplied, then values must also be supplied to the `predict` function.

`alpha` The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as

$$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1.$$

`alpha=1` is the lasso penalty, and `alpha=0` the ridge penalty.

`nlambda` The number of `lambda` values - default is 100.

`lambda.min.ratio`
Smallest value for `lambda`, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size `nobs` relative to the number of variables `nvars`. If `nobs` > `nvars`, the default is `0.0001`, close to zero. If `nobs` < `nvars`, the default is `0.01`. A very small value of `lambda.min.ratio` will lead to a saturated fit in the `nobs` < `nvars` case. This is undefined for `"binomial"` and `"multinomial"` models, and `glmnet` will exit gracefully when the percentage deviance explained is almost 1.

`lambda` A user supplied `lambda` sequence. Typical usage is to have the program compute its own `lambda` sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of `lambda` overrides this. WARNING: use with care. Avoid supplying a single value for `lambda` (for predictions after CV use `predict()` instead). Supply instead a decreasing sequence of `lambda` values. `glmnet` relies on its warms starts for speed, and its often faster to fit a whole path than compute a single fit.

`standardize` Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is `standardize=TRUE`. If variables are in the same units already, you might not wish to standardize. See details below for y standardization with `family="gaussian"`.

`intercept` Should intercept(s) be fitted (default=TRUE) or set to zero (FALSE)

`thresh` **Deprecated**. Use `control = list(thresh = ...)` or `glmnet.control(thresh = ...)` instead. Convergence threshold for coordinate descent. Factory default is `1E-7`.

`dfmax` **Deprecated**. Use `control = list(dfmax = ...)` or `glmnet.control(dfmax = ...)` instead. Limit the maximum number of variables in the model.

`pmax` **Deprecated**. Use `control = list(pmax = ...)` or `glmnet.control(pmax = ...)` instead. Limit the maximum number of variables ever to be nonzero.

`exclude` Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor for the variables excluded (next item). Users can supply instead an `exclude` function that generates the list of indices. This function is most generally defined as `function(x, y, weights, ...)`, and is called inside `glmnet` to generate the indices for excluded variables. The `...` argument is required, the others are optional. This is useful for filtering wide data, and works correctly with `cv.glmnet`. See the vignette 'Introduction' for examples.

`penalty.factor` Separate penalty factors can be applied to each coefficient. This is a number that multiplies `lambda` to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in `exclude`). Also, any `penalty.factor` that is set to `inf` is converted to an `exclude`, and then internally reset to 1. Note: the penalty factors are internally rescaled to sum to `nvars`, and the lambda sequence will reflect this change. Users can supply instead a `penalty.factor` function that generates the penalty factors. This function is most generally defined as `function(x, y, weights, ...)`, and is called inside `glmnet` to generate the penalty factors (of length `nvars`). The `...` argument is required, the others are optional. This is useful for adapting the penalty to the data, and works correctly with `cv.glmnet`.

`lower.limits` Vector of lower limits for each coefficient; default `-Inf`. Each of these must be non-positive. Can be presented as a single value (which will then be replicated), else a vector of length `nvars`

`upper.limits` Vector of upper limits for each coefficient; default `Inf`. See `lower.limits`

`maxit` **Deprecated**. Use `control = list(maxit = ...)` or `glmnet.control(maxit = ...)` instead. Maximum number of passes over the data for all lambda values; factory default is 10^5.

`type.gaussian` Two algorithm types are supported for (only) `family="gaussian"`. The default when `nvar<500` is `type.gaussian="covariance"`, and saves all inner-products ever computed. This can be much faster than `type.gaussian="naive"`, which loops through `nobs` every time an inner-product is computed. The latter can be far more efficient for `nvar >> nobs` situations, or when `nvar > 500`.

`type.logistic` If `"Newton"` then the exact hessian is used (default), while `"modified.Newton"` uses an upper-bound on the hessian, and can be faster.

`standardize.response`
This is for the `family="mgaussian"` family, and allows the user to standardize the response variables

`type.multinomial`
If `"grouped"` then a grouped lasso penalty is used on the multinomial coefficients for a variable. This ensures they are all in our out together. The default is `"ungrouped"`

`relax` If `TRUE` then for each *active set* in the path of solutions, the model is refit without any regularization. See `details` for more information. This argument is new, and users may experience convergence issues with small datasets, especially with non-gaussian families. Limiting the value of 'maxp' can alleviate these issues in some cases.

`trace.it` **Deprecated**. Use `control = list(trace.it = ...)` or `glmnet.control(trace.it = ...)` instead. If `trace.it=1`, then a progress bar is displayed.

`cox.ties` Character; the method for handling ties in Cox models. One of `"breslow"` (the current default) or `"efron"`. Applies when `family="cox"`. **The default will change to** `"efron"` **in glmnet 5.1** to match `survival::coxph`; until then, calls that do not set `cox.ties` explicitly emit a transition warning. Pass `cox.ties = "breslow"` to lock in the v5.0 default, or `cox.ties = "efron"` to preview the v5.1 behavior.

`control` A named list of algorithm control parameters, providing per-call overrides of session defaults set by `glmnet.control()`. The following keys are accepted:

- *Both execution paths*: `thresh`, `maxit`, `dfmax`, `pmax`, `trace.it`, `fdev`, `devmax`, `mnlam`, `eps`, `big`, `itrace`.
- *Core-engine path only* (ignored when `family` is a `family()` object): `pmin`, `exmx` (logistic- family kernels); `prec`, `mxit` (bounds-subsolver).
- *GLM-family (R-IRLS) path only* (ignored when `family` is a character string): `epsnr`, `mxitnr`.

Unknown keys trigger an error. Overrides are per-call and do not mutate session state (the C++-global parameters are restored to their pre-call values on exit, including on error). See `glmnet.control` for each parameter's role, scope, and factory default.

`...` Additional argument used in `relax.glmnet`. These include some of the original arguments to 'glmnet', and each must be named if used.

`fit` For `relax.glmnet` a fitted 'glmnet' object

`maxp` a limit on how many relaxed coefficients are allowed. Default is 'n-3', where 'n' is the sample size. This may not be sufficient for non-gaussian familes, in which case users should supply a smaller value. This argument can be supplied directly to 'glmnet'.

`path` Since `glmnet` does not do stepsize optimization, the Newton algorithm can get stuck and not converge, especially with relaxed fits. With `path=TRUE`, each relaxed fit on a particular set of variables is computed pathwise using the original sequence of lambda values (with a zero attached to the end). Not needed for Gaussian models, and should not be used unless needed, since will lead to longer compute times. Default is `path=FALSE`. appropriate subset of variables

`check.args` Should `relax.glmnet` make sure that all the data dependent arguments used in creating 'fit' have been resupplied. Default is 'TRUE'.

Details

The sequence of models implied by `lambda` is fit by coordinate descent. For `family="gaussian"` this is the lasso sequence if `alpha=1`, else it is the elasticnet sequence.

The objective function for `"gaussian"` is

$$1/2RSS/nobs + \lambda * penalty,$$

and for the other models it is

$$-loglik/nobs + \lambda * penalty.$$

Note also that for "gaussian", glmnet standardizes y to have unit variance (using 1/n rather than 1/(n-1) formula) before computing its lambda sequence (and then unstandardizes the resulting coefficients); if you wish to reproduce/compare results with other software, best to supply a standardized y. The coefficients for any predictor variables with zero variance are set to zero for all values of lambda.

Details on family **option:**

From version 4.0 onwards, glmnet supports both the original built-in families, as well as *any* family object as used by stats:glm(). This opens the door to a wide variety of additional models. For example family=binomial(link=cloglog) or family=negative.binomial(theta=1.5) (from the MASS library). Note that the code runs faster for the built-in families.

The built in families are specifed via a character string. For all families, the object produced is a lasso or elasticnet regularization path for fitting the generalized linear regression paths, by maximizing the appropriate penalized log-likelihood (partial likelihood for the "cox" model). Sometimes the sequence is truncated before nlambda values of lambda have been used, because of instabilities in the inverse link functions near a saturated fit. glmnet(...,family="binomial") fits a traditional logistic regression model for the log-odds. glmnet(...,family="multinomial") fits a symmetric multinomial model, where each class is represented by a linear model (on the log-scale). The penalties take care of redundancies. A two-class "multinomial" model will produce the same fit as the corresponding "binomial" model, except the pair of coefficient matrices will be equal in magnitude and opposite in sign, and half the "binomial" values. Two useful additional families are the family="mgaussian" family and the type.multinomial="grouped" option for multinomial fitting. The former allows a multi-response gaussian model to be fit, using a "group -lasso" penalty on the coefficients for each variable. Tying the responses together like this is called "multi-task" learning in some domains. The grouped multinomial allows the same penalty for the family="multinomial" model, which is also multi-responsed. For both of these the penalty on the coefficient vector for variable j is

$$(1-\alpha)/2||\beta_j||_2^2+\alpha||\beta_j||_2.$$

When alpha=1 this is a group-lasso penalty, and otherwise it mixes with quadratic just like elasticnet. A small detail in the Cox model: if death times are tied with censored times, we assume the censored times occurred just *before* the death times in computing the Breslow approximation; if users prefer the usual convention of *after*, they can add a small number to all censoring times to achieve this effect.

Details on response for family="cox"**:**

For Cox models, the response should preferably be a Surv object, created by the Surv() function in **survival** package. For right-censored data, this object should have type "right", and for (start, stop] data, it should have type "counting". To fit stratified Cox models, strata should be added to the response via the stratifySurv() function before passing the response to glmnet(). (For backward compatibility, right-censored data can also be passed as a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right censored.)

Details on relax **option:**

If relax=TRUE a duplicate sequence of models is produced, where each active set in the elasticnet path is refit without regularization. The result of this is a matching "glmnet" object which is stored on the original object in a component named "relaxed", and is part of the glmnet output.

Generally users will not call relax.glmnet directly, unless the original 'glmnet' object took a long time to fit. But if they do, they must supply the fit, and all the original arguments used to create that fit. They can limit the length of the relaxed path via 'maxp'.

Value

An object with S3 class "glmnet","*" , where "*" is "elnet", "lognet", "multnet", "fishnet" (poisson), "coxnet" or "mrelnet" for the various types of models. If the model was created with relax=TRUE then this class has a prefix class of "relaxed".

call	the call that produced this object
a0	Intercept sequence of length length(lambda)
beta	For "elnet", "lognet", "fishnet" and "coxnet" models, a nvars x length(lambda) matrix of coefficients, stored in sparse column format ("CsparseMatrix"). For "multnet" and "mgaussian", a list of nc such matrices, one for each class.
lambda	The actual sequence of lambda values used. When alpha=0, the largest lambda reported does not quite give the zero coefficients reported (lambda=inf would in principle). Instead, the largest lambda for alpha=0.001 is used, and the sequence of lambda values is derived from this.
dev.ratio	The fraction of (null) deviance explained (for "elnet", this is the R-square). The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.
nulldev	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)); The NULL model refers to the intercept model, except for the Cox, where it is the 0 model.
df	The number of nonzero coefficients for each value of lambda. For "multnet", this is the number of variables with a nonzero coefficient for *any* class.
dfmat	For "multnet" and "mrelnet" only. A matrix consisting of the number of nonzero coefficients per class
dim	dimension of coefficient matrix (ices)
nobs	number of observations
npasses	total passes over the data summed over all lambda values
offset	a logical variable indicating whether an offset was included in the model
jerr	error flag, for warnings and errors (largely for internal debugging).
relaxed	If relax=TRUE, this additional item is another glmnet object with different values for beta and dev.ratio

Author(s)

Jerome Friedman, Trevor Hastie, Balasubramanian Narasimhan, Noah Simon, Kenneth Tay and Rob Tibshirani
Maintainer: Trevor Hastie <hastie@stanford.edu>

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.
Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.
Tibshirani,Robert, Bien, J., Friedman, J., Hastie, T.,Simon, N.,Taylor, J. and Tibshirani, Ryan. (2012) *Strong Rules for Discarding Predictors in Lasso-type Problems, JRSSB, Vol. 74(2), 245-266*, https://arxiv.org/abs/1011.2234.
Hastie, T., Tibshirani, Robert and Tibshirani, Ryan (2020) *Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons, Statist. Sc. Vol. 35(4), 579-592*, https://arxiv.org/abs/1707.08692.
Glmnet webpage with four vignettes for learning how to use glmnet, documnetation for all the functions, a history of the developments, and more: https://glmnet.stanford.edu.

See Also

`print`, `predict`, `coef` and `plot` methods, and the `cv.glmnet` function.

Examples

```
# Gaussian
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = glmnet(x, y)
print(fit1)
coef(fit1, s = 0.01)  # extract coefficients at a single value of lambda
predict(fit1, newx = x[1:10, ], s = c(0.01, 0.005))  # make predictions

# Relaxed
fit1r = glmnet(x, y, relax = TRUE)  # can be used with any model

# multivariate gaussian
y = matrix(rnorm(100 * 3), 100, 3)
fit1m = glmnet(x, y, family = "mgaussian")
plot(fit1m, type.coef = "2norm")

# binomial
g2 = sample(c(0,1), 100, replace = TRUE)
fit2 = glmnet(x, g2, family = "binomial")
fit2n = glmnet(x, g2, family = binomial(link=cloglog))
fit2r = glmnet(x,g2, family = "binomial", relax=TRUE)
fit2rp = glmnet(x,g2, family = "binomial", relax=TRUE, path=TRUE)

# multinomial
g4 = sample(1:4, 100, replace = TRUE)
fit3 = glmnet(x, g4, family = "multinomial")
fit3a = glmnet(x, g4, family = "multinomial", type.multinomial = "grouped")
# poisson
N = 500
```

```
p = 20
nzc = 5
x = matrix(rnorm(N * p), N, p)
beta = rnorm(nzc)
f = x[, seq(nzc)] %*% beta
mu = exp(f)
y = rpois(N, mu)
fit = glmnet(x, y, family = "poisson")
plot(fit)
pfit = predict(fit, x, s = 0.001, type = "response")
plot(pfit, y)

# Cox
set.seed(10101)
N = 1000
p = 30
nzc = p/3
x = matrix(rnorm(N * p), N, p)
beta = rnorm(nzc)
fx = x[, seq(nzc)] %*% beta/3
hx = exp(fx)
ty = rexp(N, hx)
tcens = rbinom(n = N, prob = 0.3, size = 1)  # censoring indicator
y = cbind(time = ty, status = 1 - tcens)  # y=Surv(ty,1-tcens) with library(survival)
fit = glmnet(x, y, family = "cox")
plot(fit)

# Cox example with (start, stop] data
set.seed(2)
nobs <- 100; nvars <- 15
xvec <- rnorm(nobs * nvars)
xvec[sample.int(nobs * nvars, size = 0.4 * nobs * nvars)] <- 0
x <- matrix(xvec, nrow = nobs)
start_time <- runif(100, min = 0, max = 5)
stop_time <- start_time + runif(100, min = 0.1, max = 3)
status <- rbinom(n = nobs, prob = 0.3, size = 1)
jsurv_ss <- survival::Surv(start_time, stop_time, status)
fit <- glmnet(x, jsurv_ss, family = "cox")

# Cox example with strata
jsurv_ss2 <- stratifySurv(jsurv_ss, rep(1:2, each = 50))
fit <- glmnet(x, jsurv_ss2, family = "cox")

# Sparse
n = 10000
p = 200
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
iz = sample(1:(n * p), size = n * p * 0.85, replace = FALSE)
x[iz] = 0
sx = Matrix(x, sparse = TRUE)
inherits(sx, "sparseMatrix")  #confirm that it is sparse
beta = rnorm(nzc)
```

```
fx = x[, seq(nzc)] %*% beta
eps = rnorm(n)
y = fx + eps
px = exp(fx)
px = px/(1 + px)
ly = rbinom(n = length(px), prob = px, size = 1)
system.time(fit1 <- glmnet(sx, y))
system.time(fit2n <- glmnet(x, y))
```

`glmnet.control` *Internal glmnet algorithm parameters*

Description

View and/or change the factory default parameters that control glmnet's numerical algorithms.

Usage

```
glmnet.control(
  fdev = 1e-05,
  devmax = 0.999,
  eps = 1e-06,
  big = 9.9e+35,
  mnlam = 5,
  pmin = 1e-09,
  exmx = 250,
  prec = 1e-10,
  mxit = 100,
  itrace = 0,
  epsnr = 1e-06,
  mxitnr = 25,
  thresh = 1e-07,
  maxit = 1e+05,
  dfmax = NULL,
  pmax = NULL,
  trace.it = 0,
  factory = FALSE
)
```

Arguments

`fdev`	Path-termination parameter: minimum fractional change in deviance between consecutive lambdas for the path to continue. Scope: both paths. Factory default `1.0e-5`.
`devmax`	Path-termination parameter: maximum fraction of null deviance the path is allowed to explain before stopping. Scope: both paths. Factory default `0.999`.

`eps` Path-termination parameter: floor on `lambda.min.ratio` (smallest allowed $\lambda_{\min}/\lambda_{\max}$). Scope: both paths. Factory default `1.0e-6`.

`big` Numerical guard: finite substitute for `Inf` in `lower.limits` / `upper.limits`. Scope: both paths. Factory default `9.9e35`.

`mnlam` Path-termination parameter: minimum number of lambda values to compute before early stopping is allowed. Scope: both paths. Factory default `5`.

`pmin` Numerical guard: floor on predicted probability in logistic-family kernels (prevents `log(0)`). Implies a max of `1 - pmin`. Scope: engine only (logistic-family kernels). Factory default `1.0e-9`.

`exmx` Numerical guard: cap on `exp()` arguments in logistic-family kernels (prevents overflow). Scope: engine only (logistic-family kernels). Factory default `250`.

`prec` Bounds-subsolver convergence tolerance. Used by the engine's `bnorm()` subroutine when a coefficient has a finite `lower` or `upper` bound. Scope: engine only (bounds-constrained fits). Factory default `1.0e-10`.

`mxit` Bounds-subsolver iteration budget. Paired with `prec`. Scope: engine only (bounds-constrained fits). Factory default `100`.

`itrace` Progress-bar flag. `1` enables the progress bar in `glmnet()` and `cv.glmnet()`. Scope: both paths. Aliased with `trace.it` (setting one sets the other). Factory default `0`.

`epsnr` Newton-Raphson convergence tolerance for the R-level IRLS loop in `glmnet.fit()`. **Scope: R-IRLS path only** — the core-engine kernels do not read this parameter. Factory default `1.0e-6`.

`mxitnr` Newton-Raphson iteration budget for the R-level IRLS loop in `glmnet.fit()`. **Scope: R-IRLS path only**. Factory default `25`.

`thresh` Inner coordinate-descent convergence threshold. Each inner CD loop continues until the maximum change in the objective after any coefficient update is less than `thresh` times the null deviance. Scope: both paths. Factory default `1e-7`. Can also be set per-call via `control = list(thresh = ...)` in `glmnet()`.

`maxit` Inner coordinate-descent iteration budget: maximum total passes over the data across all lambda values. Scope: both paths. Factory default `100000`. Can also be set per-call via `control = list(maxit = ...)`.

`dfmax` Coefficient-count cap: limit the maximum number of variables in the model at any lambda. Scope: both paths. Factory default `NULL`, which resolves at call time to `nvars + 1` (i.e., unconstrained). Can also be set per-call via `control = list(dfmax = ...)`.

`pmax` Coefficient-count cap: limit the maximum number of variables ever to be nonzero anywhere on the path. Scope: both paths. Factory default `NULL`, which resolves at call time to `min(dfmax * 2 + 20, nvars)`. Can also be set per-call via `control = list(pmax = ...)`.

`trace.it` Progress-bar flag (alias of `itrace`). Scope: both paths. Factory default `0`.

`factory` If `TRUE`, reset all parameters to their factory defaults. Default is `FALSE`.

Details

If called with no arguments, `glmnet.control()` returns a list with the current settings of these parameters. Any arguments included in the call set those parameters to the new values, and then silently return the updated settings. Values set via `glmnet.control()` persist for the duration of the R session. All parameters listed here can also be supplied per-call via the `control = list(...)` argument of `glmnet()`, which does *not* mutate the session state; see that function's documentation for the precedence rules.

Value

A list with named elements as in the argument list.

Parameter taxonomy

glmnet has two execution paths, and not every control parameter is consumed by both:

- **Core-engine path** — `family` passed as a character string (`"gaussian"`, `"binomial"`, `"poisson"`, `"multinomial"`, `"mgaussian"`, `"cox"`) routes to purpose-built C++ glmnetpp kernels.
- **GLM-family (R-IRLS) path** — `family` passed as a `family()` object (or `glmnet.path()` / `glmnet.fit()` called directly) runs an IRLS loop implemented in R that uses the C++ `wls` kernel as its inner solver.

The table below shows the role and the scope of each parameter. "Scope: engine" means the parameter is only meaningful on the core-engine path; "Scope: R-IRLS" means only on the GLM-family path; "Scope: both" means consulted by both.

Parameter	Role	Scope	Notes
`fdev`	path termination	both	C++ name `sml`; min fractional deviance change
`devmax`	path termination	both	C++ name `rsqmax`; max explained-deviance ratio
`mnlam`	path termination	both	min lambda count before early stop
`eps`	path termination	both	lambda.min.ratio floor
`thresh`	CD tolerance	both	inner coordinate-descent convergence
`maxit`	CD budget	both	max CD passes across all lambdas
`dfmax`	coefficient-count cap	both	default resolves to `nvars + 1` at call time
`pmax`	coefficient-count cap	both	default resolves to `min(dfmax*2+20, nvars)` at call time
`big`	numerical guard	both	`Inf` substitute for bounds limits
`itrace`	progress	both	aliased with `trace.it`
`trace.it`	progress	both	aliased with `itrace`
`pmin`	probability floor	engine	logistic-family kernels only (`lognet`, `multnet`)
`exmx`	exp() cap	engine	logistic-family kernels only
`prec`	bounds-subsolver tolerance	engine	active when a coefficient has finite `lower` / `upper`
`mxit`	bounds-subsolver budget	engine	paired with `prec` via C++ `chg_bnorm()`
`epsnr`	Newton-Raphson tolerance	R-IRLS	*inert on the core-engine path*; only `glmnet.fit()` reads it
`mxitnr`	Newton-Raphson budget	R-IRLS	*inert on the core-engine path*; only `glmnet.fit()` reads it

The naming zoo (three tolerances, three iteration budgets) reflects the fact that glmnet contains three nested loops:

- **Outer path**: iterate over lambda values. Early termination governed by `fdev`, `devmax`, `mnlam`, `eps`.
- **Middle Newton/IRLS loop** (only on the R-IRLS path): convergence by `epsnr`, budget by `mxitnr`.
- **Inner coordinate descent**: convergence by `thresh`, budget by `maxit`.
- **Bounds subsolver** (only for coefficients with finite lower/upper): convergence by `prec`, budget by `mxit`.

Author(s)

Jerome Friedman, Kenneth Tay, Trevor Hastie
Maintainer: Trevor Hastie <hastie@stanford.edu>

See Also

glmnet

Examples

```
glmnet.control(fdev = 0)  # continue along path even though not much changes
glmnet.control(thresh = 1e-8)  # tighten CD convergence for session
glmnet.control()  # view current settings
glmnet.control(factory = TRUE)  # reset all the parameters to their default
```

---

glmnet.fit *Fit a GLM with elastic net regularization for a single value of lambda*

Description

Fit a generalized linear model via penalized maximum likelihood for a single value of lambda. Can deal with any GLM family.

Usage

```
glmnet.fit(
  x,
  y,
  weights,
  lambda,
  alpha = 1,
  offset = rep(0, nobs),
  family = gaussian(),
  intercept = TRUE,
  penalty.factor = rep(1, nvars),
  exclude = c(),
  lower.limits = -Inf,
```

```
  upper.limits = Inf,
  warm = NULL,
  from.glmnet.path = FALSE,
  save.fit = FALSE,
  control = glmnet.control()
)
```

Arguments

x	Input matrix, of dimension nobs x nvars; each row is an observation vector. If it is a sparse matrix, it is assumed to be unstandardized. It should have attributes xm and xs, where xm(j) and xs(j) are the centering and scaling factors for variable j respsectively. If it is not a sparse matrix, it is assumed that any standardization needed has already been done.
y	Quantitative response variable.
weights	Observation weights. glmnet.fit does NOT standardize these weights.
lambda	A single value for the lambda hyperparameter.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1.$$ alpha=1 is the lasso penalty, and alpha=0 the ridge penalty.
offset	A vector of length nobs that is included in the linear predictor. Useful for the "poisson" family (e.g. log of exposure time), or for refining a model by starting at a current fit. Default is NULL. If supplied, then values must also be supplied to the predict function.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is gaussian(). (See family for details on family functions.)
intercept	Should intercept be fitted (default=TRUE) or set to zero (FALSE)?
penalty.factor	Separate penalty factors can be applied to each coefficient. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude). Note: the penalty factors are internally rescaled to sum to nvars.
exclude	Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor.
lower.limits	Vector of lower limits for each coefficient; default -Inf. Each of these must be non-positive. Can be presented as a single value (which will then be replicated), else a vector of length nvars.
upper.limits	Vector of upper limits for each coefficient; default Inf. See lower.limits.
warm	Either a glmnetfit object or a list (with names beta and a0 containing coefficients and intercept respectively) which can be used as a warm start. Default is NULL, indicating no warm start. For internal use only.

`from.glmnet.path`
: Was `glmnet.fit()` called from `glmnet.path()`? Default is FALSE.This has implications for computation of the penalty factors.

`save.fit`
: Return the warm start object? Default is FALSE.

`control`
: A fully resolved 17-key control list of the form returned by `glmnet.control()`. Default is `glmnet.control()` – current session state. This function does not resolve or validate the list; keys (`thresh`, `maxit`, `trace.it`, `epsnr`, `mxitnr`, `big`, etc.) are read directly. See `?glmnet.path` for the same contract.

Details

WARNING: Users should not call `glmnet.fit` directly. Higher-level functions in this package call `glmnet.fit` as a subroutine. If a warm start object is provided, some of the other arguments in the function may be overriden.

`glmnet.fit` solves the elastic net problem for a single, user-specified value of lambda. `glmnet.fit` works for any GLM family. It solves the problem using iteratively reweighted least squares (IRLS). For each IRLS iteration, `glmnet.fit` makes a quadratic (Newton) approximation of the log-likelihood, then calls `elnet.fit` to minimize the resulting approximation.

In terms of standardization: `glmnet.fit` does not standardize `x` and `weights`. `penalty.factor` is standardized so that they sum up to `nvars`.

Value

An object with class "glmnetfit" and "glmnet". The list returned contains more keys than that of a "glmnet" object.

`a0`
: Intercept value.

`beta`
: A `nvars` x `1` matrix of coefficients, stored in sparse matrix format.

`df`
: The number of nonzero coefficients.

`dim`
: Dimension of coefficient matrix.

`lambda`
: Lambda value used.

`dev.ratio`
: The fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.

`nulldev`
: Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)). The null model refers to the intercept model.

`npasses`
: Total passes over the data.

`jerr`
: Error flag, for warnings and errors (largely for internal debugging).

`offset`
: A logical variable indicating whether an offset was included in the model.

`call`
: The call that produced this object.

`nobs`
: Number of observations.

`warm_fit`
: If `save.fit=TRUE`, output of C++ routine, used for warm starts. For internal use only.

family	Family used for the model.
converged	A logical variable: was the algorithm judged to have converged?
boundary	A logical variable: is the fitted value on the boundary of the attainable values?
obj_function	Objective function value at the solution.

`glmnet.measures` *Display the names of the measures used in CV for different "glmnet" families*

Description

Produces a list of names of measures

Usage

```
glmnet.measures(
 family = c("all", "gaussian", "binomial", "poisson", "multinomial", "cox", "mgaussian",
    "GLM")
)
```

Arguments

family	If a "glmnet" family is supplied, a list of the names of measures available for that family are produced. Default is "all", in which case the names of measures for all families are produced.

Details

Try it and see. A very simple function to provide information

Author(s)

Trevor Hastie
Maintainer: Trevor Hastie <hastie@stanford.edu>

See Also

`cv.glmnet` and `assess.glmnet`.

`glmnet.path` *Fit a GLM with elastic net regularization for a path of lambda values*

Description

Fit a generalized linear model via penalized maximum likelihood for a path of lambda values. Can deal with any GLM family.

Usage

```
glmnet.path(
  x,
  y,
  weights = NULL,
  lambda = NULL,
  nlambda = 100,
  lambda.min.ratio = ifelse(nobs < nvars, 0.01, 1e-04),
  alpha = 1,
  offset = NULL,
  family = gaussian(),
  standardize = TRUE,
  intercept = TRUE,
  penalty.factor = rep(1, nvars),
  exclude = integer(0),
  lower.limits = -Inf,
  upper.limits = Inf,
  control = glmnet.control()
)
```

Arguments

`x`	Input matrix, of dimension `nobs` x `nvars`; each row is an observation vector. Can be a sparse matrix.
`y`	Quantitative response variable.
`weights`	Observation weights. Default is 1 for each observation.
`lambda`	A user supplied lambda sequence. Typical usage is to have the program compute its own lambda sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of lambda overrides this.
`nlambda`	The number of lambda values, default is 100.
`lambda.min.ratio`	Smallest value for lambda as a fraction of lambda.max, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size `nobs` relative to the number of variables `nvars`. If `nobs` >= `nvars`, the default is 0.0001, close to zero. If `nobs` < `nvars`, the default is 0.01. A very small value of `lambda.min.ratio` will lead to a saturated fit in the `nobs` < `nvars` case. This is undefined for some families of models, and the

function will exit gracefully when the percentage deviance explained is almost 1.

`alpha` The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as

$$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1.$$

`alpha=1` is the lasso penalty, and `alpha=0` the ridge penalty.

`offset` A vector of length `nobs` that is included in the linear predictor. Useful for the "poisson" family (e.g. log of exposure time), or for refining a model by starting at a current fit. Default is NULL. If supplied, then values must also be supplied to the `predict` function.

`family` A description of the error distribution and link function to be used in the model. This is the result of a call to a family function. Default is `gaussian()`. (See `family` for details on family functions.)

`standardize` Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is `standardize=TRUE`. If variables are in the same units already, you might not wish to standardize.

`intercept` Should intercept be fitted (default=TRUE) or set to zero (FALSE)?

`penalty.factor` Separate penalty factors can be applied to each coefficient. This is a number that multiplies `lambda` to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables (and implicitly infinity for variables listed in exclude). Note: the penalty factors are internally rescaled to sum to `nvars`.

`exclude` Indices of variables to be excluded from the model. Default is none. Equivalent to an infinite penalty factor.

`lower.limits` Vector of lower limits for each coefficient; default `-Inf`. Each of these must be non-positive. Can be presented as a single value (which will then be replicated), else a vector of length `nvars`.

`upper.limits` Vector of upper limits for each coefficient; default `Inf`. See `lower.limits`.

`control` A fully resolved 17-key control list of the form returned by `glmnet.control()`. Default is `glmnet.control()` – i.e., the current session state. This function does not resolve, validate, or layer the list; it reads keys (`thresh`, `maxit`, `trace.it`, `fdev`, `eps`, `epsnr`, `mxitnr`, etc.) from it directly. When called from `glmnet()`, the argument is populated by `.resolve_control()` and already reflects any per-call overrides; see `glmnet.control` for the parameter taxonomy. When calling this function directly (e.g., from test code), either pass nothing (use session state) or build a full list via `modifyList(glmnet.control(), ...)`.

Details

`glmnet.path` solves the elastic net problem for a path of lambda values. It generalizes `glmnet::glmnet` in that it works for any GLM family.

Sometimes the sequence is truncated before `nlambda` values of lambda have been used. This happens when `glmnet.path` detects that the decrease in deviance is marginal (i.e. we are near a saturated fit).

Value

An object with class "glmnetfit" and "glmnet".

a0	Intercept sequence of length length(lambda).
beta	A nvars x length(lambda) matrix of coefficients, stored in sparse matrix format.
df	The number of nonzero coefficients for each value of lambda.
dim	Dimension of coefficient matrix.
lambda	The actual sequence of lambda values used. When alpha=0, the largest lambda reported does not quite give the zero coefficients reported (lambda=inf would in principle). Instead, the largest lambda for alpha=0.001 is used, and the sequence of lambda values is derived from this.
dev.ratio	The fraction of (null) deviance explained. The deviance calculations incorporate weights if present in the model. The deviance is defined to be 2*(loglike_sat - loglike), where loglike_sat is the log-likelihood for the saturated model (a model with a free parameter per observation). Hence dev.ratio=1-dev/nulldev.
nulldev	Null deviance (per observation). This is defined to be 2*(loglike_sat -loglike(Null)). The null model refers to the intercept model.
npasses	Total passes over the data summed over all lambda values.
jerr	Error flag, for warnings and errors (largely for internal debugging).
offset	A logical variable indicating whether an offset was included in the model.
call	The call that produced this object.
family	Family used for the model.
nobs	Number of observations.

Examples

```
set.seed(1)
x <- matrix(rnorm(100 * 20), nrow = 100)
y <- ifelse(rnorm(100) > 0, 1, 0)

# binomial with probit link
fit1 <- glmnet:::glmnet.path(x, y, family = binomial(link = "probit"))
```

---

makeX *convert a data frame to a data matrix with one-hot encoding*

Description

Converts a data frame to a data matrix suitable for input to glmnet. Factors are converted to dummy matrices via "one-hot" encoding. Options deal with missing values and sparsity.

Usage

```
makeX(train, test = NULL, na.impute = FALSE, sparse = FALSE, ...)
```

Arguments

train	Required argument. A dataframe consisting of vectors, matrices and factors
test	Optional argument. A dataframe matching 'train' for use as testing data
na.impute	Logical, default FALSE. If TRUE, missing values for any column in the resultant 'x' matrix are replaced by the means of the nonmissing values derived from 'train'
sparse	Logical, default FALSE. If TRUE then the returned matrice(s) are converted to matrices of class "CsparseMatrix". Useful if some factors have a large number of levels, resulting in very big matrices, mostly zero
...	additional arguments, currently unused

Details

The main function is to convert factors to dummy matrices via "one-hot" encoding. Having the 'train' and 'test' data present is useful if some factor levels are missing in either. Since a factor with k levels leads to a submatrix with 1/k entries zero, with large k the sparse=TRUE option can be helpful; a large matrix will be returned, but stored in sparse matrix format. Finally, the function can deal with missing data. The current version has the option to replace missing observations with the mean from the training data. For dummy submatrices, these are the mean proportions at each level.

Value

If only 'train' was provided, the function returns a matrix 'x'. If missing values were imputed, this matrix has an attribute containing its column means (before imputation). If 'test' was provided as well, a list with two components is returned: 'x' and 'xtest'.

Author(s)

Trevor Hastie
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

glmnet

Examples

```
set.seed(101)
### Single data frame
X = matrix(rnorm(20), 10, 2)
X3 = sample(letters[1:3], 10, replace = TRUE)
X4 = sample(LETTERS[1:3], 10, replace = TRUE)
df = data.frame(X, X3, X4)
makeX(df)
makeX(df, sparse = TRUE)

### Single data freame with missing values
Xn = X
Xn[3, 1] = NA
```

```
Xn[5, 2] = NA
X3n = X3
X3n[6] = NA
X4n = X4
X4n[9] = NA
dfn = data.frame(Xn, X3n, X4n)

makeX(dfn)
makeX(dfn, sparse = TRUE)
makeX(dfn, na.impute = TRUE)
makeX(dfn, na.impute = TRUE, sparse = TRUE)

### Test data as well
X = matrix(rnorm(10), 5, 2)
X3 = sample(letters[1:3], 5, replace = TRUE)
X4 = sample(LETTERS[1:3], 5, replace = TRUE)
dft = data.frame(X, X3, X4)

makeX(df, dft)
makeX(df, dft, sparse = TRUE)

### Missing data in test as well
Xn = X
Xn[3, 1] = NA
Xn[5, 2] = NA
X3n = X3
X3n[1] = NA
X4n = X4
X4n[2] = NA
dftn = data.frame(Xn, X3n, X4n)

makeX(dfn, dftn)
makeX(dfn, dftn, sparse = TRUE)
makeX(dfn, dftn, na.impute = TRUE)
makeX(dfn, dftn, sparse = TRUE, na.impute = TRUE)
```

`MultiGaussianExample` *Synthetic dataset with multiple Gaussian responses*

Description

Randomly generated data for multi-response Gaussian regression example.

Usage

```
data(MultiGaussianExample)
```

Format

List containing the following elements:

x 100 by 20 matrix of numeric values.

y 100 by 4 matrix of numeric values, each column representing one response vector.

MultinomialExample *Synthetic dataset with multinomial response*

Description

Randomly generated data for multinomial regression example.

Usage

```
data(MultinomialExample)
```

Format

List containing the following elements:

x 500 by 30 matrix of numeric values.

y Numeric vector of length 500 containing 142 ones, 174 twos and 184 threes.

---

mycoxph *Helper function to fit coxph model for survfit.coxnet*

Description

This function constructs the coxph call needed to run the "hack" of coxph with 0 iterations. It's a separate function as we have to deal with function options like strata, offset and observation weights.

Usage

```
mycoxph(object, s, ...)
```

Arguments

object	A class coxnet object.
s	The value of the penalty parameter lambda at which the survival curve is required.
...	The same ... that was passed to survfit.coxnet.

mycoxpred *Helper function to amend ... for new data in survfit.coxnet*

Description

This function amends the function arguments passed to survfit.coxnet via ... if new data was passed to survfit.coxnet. It's a separate function as we have to deal with function options like newstrata and newoffset.

Usage

```
mycoxpred(object, s, ...)
```

Arguments

object	A class coxnet object.
s	The response for the fitted model.
...	The same ... that was passed to survfit.coxnet.

na.replace *Replace the missing entries in a matrix columnwise with the entries in a supplied vector*

Description

Missing entries in any given column of the matrix are replaced by the column means or the values in a supplied vector.

Usage

```
na.replace(x, m = colMeans(x, na.rm = TRUE))
```

Arguments

x	A matrix with potentially missing values, and also potentially in sparse matrix format (i.e. inherits from "sparseMatrix")
m	Optional argument. A vector of values used to replace the missing entries, columnwise. If missing, the column means of 'x' are used

Details

This is a simple imputation scheme. This function is called by makeX if the na.impute=TRUE option is used, but of course can be used on its own. If 'x' is sparse, the result is sparse, and the replacements are done so as to maintain sparsity.

Value

A version of 'x' is returned with the missing values replaced.

Author(s)

Trevor Hastie
Maintainer: Trevor Hastie hastie@stanford.edu

See Also

`makeX` and `glmnet`

Examples

```
set.seed(101)
### Single data frame
X = matrix(rnorm(20), 10, 2)
X[3, 1] = NA
X[5, 2] = NA
X3 = sample(letters[1:3], 10, replace = TRUE)
X3[6] = NA
X4 = sample(LETTERS[1:3], 10, replace = TRUE)
X4[9] = NA
dfn = data.frame(X, X3, X4)

x = makeX(dfn)
m = rowSums(x, na.rm = TRUE)
na.replace(x, m)

x = makeX(dfn, sparse = TRUE)
na.replace(x, m)
```

---

obj_function *Elastic net objective function value*

Description

Returns the elastic net objective function value.

Usage

```
obj_function(y, mu, weights, family, lambda, alpha, coefficients, vp)
```

Arguments

y	Quantitative response variable.
mu	Model's predictions for y.
weights	Observation weights.
family	A description of the error distribution and link function to be used in the model. This is the result of a call to a family function.
lambda	A single value for the lambda hyperparameter.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$.
coefficients	The model's coefficients (excluding intercept).
vp	Penalty factors for each of the coefficients.

---

pen_function *Elastic net penalty value*

Description

Returns the elastic net penalty value without the lambda factor.

Usage

```
pen_function(coefficients, alpha = 1, vp = 1)
```

Arguments

coefficients	The model's coefficients (excluding intercept).
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$.
vp	Penalty factors for each of the coefficients.

Details

The penalty is defined as

$$(1-\alpha)/2 \sum vp_j \beta_j^2 + \alpha \sum vp_j |\beta|.$$

Note the omission of the multiplicative lambda factor.

plot.cv.glmnet *plot the cross-validation curve produced by cv.glmnet*

Description

Plots the cross-validation curve, and upper and lower standard deviation curves, as a function of the lambda values used. If the object has class "cv.relaxed" a different plot is produced, showing both lambda and gamma

Usage

```
## S3 method for class 'cv.glmnet'
plot(x, sign.lambda = -1, ...)

## S3 method for class 'cv.relaxed'
plot(x, se.bands = TRUE, sign.lambda = -1, ...)
```

Arguments

x	fitted "cv.glmnet" object
sign.lambda	Either plot against log(lambda) or its negative if sign.lambda=-1 (default).
...	Other graphical parameters to plot
se.bands	Should shading be produced to show standard-error bands; default is TRUE

Details

A plot is produced, and nothing is returned.

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent*

See Also

glmnet and cv.glmnet.

Examples

```
set.seed(1010)
n = 1000
p = 100
nzc = trunc(p/10)
x = matrix(rnorm(n * p), n, p)
beta = rnorm(nzc)
fx = (x[, seq(nzc)] %*% beta)
eps = rnorm(n) * 5
y = drop(fx + eps)
px = exp(fx)
px = px/(1 + px)
ly = rbinom(n = length(px), prob = px, size = 1)
cvob1 = cv.glmnet(x, y)
plot(cvob1)
title("Gaussian Family", line = 2.5)
cvob1r = cv.glmnet(x, y, relax = TRUE)
plot(cvob1r)
frame()
set.seed(1011)
par(mfrow = c(2, 2), mar = c(4.5, 4.5, 4, 1))
cvob2 = cv.glmnet(x, ly, family = "binomial")
plot(cvob2)
title("Binomial Family", line = 2.5)
## set.seed(1011)
## cvob3 = cv.glmnet(x, ly, family = "binomial", type = "class")
## plot(cvob3)
## title("Binomial Family", line = 2.5)
```

`plot.glmnet` *plot coefficients from a "glmnet" object*

Description

Produces a coefficient profile plot of the coefficient paths for a fitted `"glmnet"` object.

Usage

```
## S3 method for class 'glmnet'
plot(
  x,
  xvar = c("lambda", "norm", "dev"),
  label = FALSE,
  sign.lambda = -1,
  ...
)

## S3 method for class 'mrelnet'
```

```
plot(
  x,
  xvar = c("lambda", "norm", "dev"),
  label = FALSE,
  sign.lambda = -1,
  type.coef = c("coef", "2norm"),
  ...
)

## S3 method for class 'multnet'
plot(
  x,
  xvar = c("lambda", "norm", "dev"),
  label = FALSE,
  sign.lambda = -1,
  type.coef = c("coef", "2norm"),
  ...
)

## S3 method for class 'relaxed'
plot(
  x,
  xvar = c("lambda", "dev"),
  label = FALSE,
  sign.lambda = -1,
  gamma = 1,
  ...
)
```

Arguments

x	fitted "glmnet" model
xvar	What is on the X-axis. "lambda" plots against the log-lambda sequence, "norm" against the L1-norm of the coefficients, and "dev" against the percent deviance explained. Warning: "norm" is the L1 norm of the coefficients on the glmnet object. There are many reasons why this might not be appropriate, such as automatic standardization, penalty factors, and values of alpha less than 1, which can lead to unusual looking plots.
label	If TRUE, label the curves with variable sequence numbers.
sign.lambda	If xvar="lambda" and sign.lambda=1 then we plot against log(lambda); if sign.lambda=-1 (default) we plot against -log(lambda).
...	Other graphical parameters to plot
type.coef	If type.coef="2norm" then a single curve per variable, else if type.coef="coef", a coefficient plot per response
gamma	Value of the mixing parameter for a "relaxed" fit

Details

A coefficient profile plot is produced. If x is a multinomial model, a coefficient plot is produced for each class.

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent*

See Also

`glmnet`, and `print`, `predict` and `coef` methods.

Examples

```
x=matrix(rnorm(100*20),100,20)
y=rnorm(100)
g2=sample(1:2,100,replace=TRUE)
g4=sample(1:4,100,replace=TRUE)
fit1=glmnet(x,y)
plot(fit1)
plot(fit1,xvar="lambda",label=TRUE)
fit3=glmnet(x,g4,family="multinomial")
plot(fit3,pch=19)
```

PoissonExample *Synthetic dataset with count response*

Description

Randomly generated data for Poisson regression example.

Usage

```
data(PoissonExample)
```

Format

List containing the following elements:

x 500 by 20 matrix of numeric values.

y Numeric vector of length 500 consisting of non-negative integers.

predict.cv.glmnet *make predictions from a "cv.glmnet" object.*

Description

This function makes predictions from a cross-validated glmnet model, using the stored "glmnet.fit" object, and the optimal value chosen for lambda (and gamma for a 'relaxed' fit.

Usage

```
## S3 method for class 'cv.glmnet'
predict(object, newx, s = c("lambda.1se", "lambda.min"), ...)

## S3 method for class 'cv.relaxed'
predict(
  object,
  newx,
  s = c("lambda.1se", "lambda.min"),
  gamma = c("gamma.1se", "gamma.min"),
  ...
)
```

Arguments

object	Fitted "cv.glmnet" or "cv.relaxed" object.
newx	Matrix of new values for x at which predictions are to be made. Must be a matrix; can be sparse as in Matrix package. See documentation for predict.glmnet.
s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value s="lambda.1se" stored on the CV object. Alternatively s="lambda.min" can be used. If s is numeric, it is taken as the value(s) of lambda to be used. (For historical reasons we use the symbol 's' rather than 'lambda' to reference this parameter)
...	Not used. Other arguments to predict.
gamma	Value (single) of 'gamma' at which predictions are to be made

Details

This function makes it easier to use the results of cross-validation to make a prediction.

Value

The object returned depends on the ... argument which is passed on to the predict method for glmnet objects.

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent (2010), Journal of Statistical Software, Vol. 33(1), 1-22*, doi:10.18637/jss.v033.i01.
Simon, N., Friedman, J., Hastie, T. and Tibshirani, R. (2011) *Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent, Journal of Statistical Software, Vol. 39(5), 1-13*, doi:10.18637/jss.v039.i05.
Hastie, T., Tibshirani, Robert and Tibshirani, Ryan (2020) *Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons, Statist. Sc. Vol. 35(4), 579-592*, https://arxiv.org/abs/1707.08692.
Glmnet webpage with four vignettes, https://glmnet.stanford.edu.

See Also

`glmnet`, and `print`, and `coef` methods, and `cv.glmnet`.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
cv.fit = cv.glmnet(x, y)
predict(cv.fit, newx = x[1:5, ])
coef(cv.fit)
coef(cv.fit, s = "lambda.min")
predict(cv.fit, newx = x[1:5, ], s = c(0.001, 0.002))
cv.fitr = cv.glmnet(x, y, relax = TRUE)
predict(cv.fit, newx = x[1:5, ])
coef(cv.fit)
coef(cv.fit, s = "lambda.min", gamma = "gamma.min")
predict(cv.fit, newx = x[1:5, ], s = c(0.001, 0.002), gamma = "gamma.min")
```

`predict.glmnetfit` *Get predictions from a* `glmnetfit` *fit object*

Description

Gives fitted values, linear predictors, coefficients and number of non-zero coefficients from a fitted `glmnetfit` object.

Usage

```
## S3 method for class 'glmnetfit'
predict(
  object,
  newx,
  s = NULL,
  type = c("link", "response", "coefficients", "nonzero"),
  exact = FALSE,
  newoffset,
  ...
)
```

Arguments

object	Fitted "glmnetfit" object.
newx	Matrix of new values for x at which predictions are to be made. Must be a matrix. This argument is not used for type = c("coefficients","nonzero").
s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model.
type	Type of prediction required. Type "link" gives the linear predictors (eta scale); Type "response" gives the fitted values (mu scale). Type "coefficients" computes the coefficients at the requested values for s. Type "nonzero" returns a list of the indices of the nonzero coefficients for each value of s.
exact	This argument is relevant only when predictions are made at values of s (lambda) *different* from those used in the fitting of the original model. If exact=FALSE (default), then the predict function uses linear interpolation to make predictions for values of s (lambda) that do not coincide with those used in the fitting algorithm. While this is often a good approximation, it can sometimes be a bit coarse. With exact=TRUE, these different values of s are merged (and sorted) with object$lambda, and the model is refit before predictions are made. In this case, it is required to supply the original data x= and y= as additional named arguments to predict() or coef(). The workhorse predict.glmnet() needs to update the model, and so needs the data used to create it. The same is true of weights, offset, penalty.factor, lower.limits, upper.limits if these were used in the original call. Failure to do so will result in an error.
newoffset	If an offset is used in the fit, then one must be supplied for making predictions (except for type="coefficients" or type="nonzero").
...	This is the mechanism for passing arguments like x= when exact=TRUE; see exact argument.

Value

The object returned depends on type.

print.cv.glmnet *print a cross-validated glmnet object*

Description

Print a summary of the results of cross-validation for a glmnet model.

Usage

```
## S3 method for class 'cv.glmnet'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

x	fitted 'cv.glmnet' object
digits	significant digits in printout
...	additional print arguments

Details

A summary of the cross-validated fit is produced, slightly different for a 'cv.relaxed' object than for a 'cv.glmnet' object. Note that a 'cv.relaxed' object inherits from class 'cv.glmnet', so by directly invoking `print.cv.glmnet(object)` will print the summary as if `relax=TRUE` had not been used.

Author(s)

Jerome Friedman, Trevor Hastie and Rob Tibshirani
Maintainer: Trevor Hastie hastie@stanford.edu

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008) *Regularization Paths for Generalized Linear Models via Coordinate Descent*
https://arxiv.org/abs/1707.08692
Hastie, T., Tibshirani, Robert, Tibshirani, Ryan (2019) *Extended Comparisons of Best Subset Selection, Forward Stepwise Selection, and the Lasso*

See Also

`glmnet`, `predict` and `coef` methods.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = cv.glmnet(x, y)
print(fit1)
fit1r = cv.glmnet(x, y, relax = TRUE)
print(fit1r)
## print.cv.glmnet(fit1r)  ## CHECK WITH TREVOR
```

`print.glmnet` *print a glmnet object*

Description

Print a summary of the glmnet path at each step along the path.

Usage

```
## S3 method for class 'glmnet'
print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`x`	fitted glmnet object
`digits`	significant digits in printout
`...`	additional print arguments

Details

The call that produced the object `x` is printed, followed by a three-column matrix with columns `Df`, `%Dev` and `Lambda`. The `Df` column is the number of nonzero coefficients (Df is a reasonable name only for lasso fits). `%Dev` is the percent deviance explained (relative to the null deviance). In the case of a 'relaxed' fit, an additional column is inserted, `%Dev R` which gives the percent deviance explained by the relaxed model. For a "bigGlm" model, a simpler summary is printed.

Value

The matrix above is silently returned

References

Friedman, J., Hastie, T. and Tibshirani, R. (2008). Regularization Paths for Generalized Linear Models via Coordinate Descent

See Also

`glmnet`, `predict` and `coef` methods.

Examples

```
x = matrix(rnorm(100 * 20), 100, 20)
y = rnorm(100)
fit1 = glmnet(x, y)
print(fit1)
```

QuickStartExample *Synthetic dataset with Gaussian response*

Description

Randomly generated data for Gaussian regression example.

Usage

```
data(QuickStartExample)
```

Format

List containing the following elements:

x 100 by 20 matrix of numeric values.

y Numeric vector of length 100.

---

response.coxnet *Make response for coxnet*

Description

Internal function to make the response y passed to glmnet suitable for coxnet (i.e. glmnet with family = "cox"). Sanity checks are performed here too.

Usage

```
response.coxnet(y)
```

Arguments

y	Response variable. Either a class "Surv" object or a two-column matrix with columns named 'time' and 'status'.

Details

If y is a class "Surv" object, this function returns y with no changes. If y is a two-column matrix with columns named 'time' and 'status', it is converted into a "Surv" object.

Value

A class "Surv" object.

rmult *Generate multinomial samples from a probability matrix*

Description

Generate multinomial samples

Usage

```
rmult(p)
```

Arguments

p	matrix of probabilities, with number of columns the number of classes

Details

Simple function that calls the `rmultinom` function. It generates a class label for each row of its input matrix of class probabilities.

Value

a vector of class memberships

Author(s)

Trevor Hastie
Maintainer: Trevor Hastie hastie@stanford.edu

SparseExample *Synthetic dataset with sparse design matrix*

Description

Randomly generated data for Gaussian regression example with the design matrix x being in sparse matrix format.

Usage

```
data(SparseExample)
```

Format

List containing the following elements:

x 100 by 20 matrix of numeric values. x is in sparse matrix format, having class "dgCMatrix".

y Numeric vector of length 100.

stratifySurv *Add strata to a Surv object*

Description

Helper function to add strata as an attribute to a Surv object. The output of this function can be used as the response in `glmnet()` for fitting stratified Cox models.

Usage

```
stratifySurv(y, strata = rep(1, length(y)))
```

Arguments

`y`	A Surv object.
`strata`	A vector of length equal to the number of observations in y, indicating strata membership. Default is all belong to same strata.

Details

When fitting a stratified Cox model with `glmnet()`, strata should be added to a `Surv` response with this helper function. Note that it is not sufficient to add strata as an attribute to the `Surv` response manually: if the result does not have class `stratifySurv`, subsetting of the response will not work properly.

Value

An object of class `stratifySurv` (in addition to all the classes `y` belonged to).

Examples

```
y <- survival::Surv(1:10, rep(0:1, length.out = 10))
strata <- rep(1:3, length.out = 10)
y2 <- stratifySurv(y, strata)  # returns stratifySurv object
```

survfit.coxnet *Compute a survival curve from a coxnet object*

Description

Computes the predicted survivor function for a Cox proportional hazards model with elastic net penalty.

Usage

```
## S3 method for class 'coxnet'
survfit(formula, s = NULL, ...)
```

Arguments

formula	A class coxnet object.
s	Value(s) of the penalty parameter lambda at which the survival curve is required. Default is the entire sequence used to create the model. However, it is recommended that survfit.coxnet is called for a single penalty parameter.
...	This is the mechanism for passing additional arguments like (i) x= and y= for the x and y used to fit the model, (ii) weights= and offset= when the model was fit with these options, (iii) arguments for new data (newx, newoffset, newstrata), and (iv) arguments to be passed to survfit.coxph().

Details

To be consistent with other functions in glmnet, if s is not specified, survival curves are returned for the entire lambda sequence. This is not recommended usage: it is best to call survfit.coxnet with a single value of the penalty parameter for the s option.

Value

If s is a single value, an object of class "survfitcox" and "survfit" containing one or more survival curves. Otherwise, a list of such objects, one element for each value in s. Methods defined for survfit objects are print, summary and plot.

Examples

```
set.seed(2)
nobs <- 100; nvars <- 15
xvec <- rnorm(nobs * nvars)
xvec[sample.int(nobs * nvars, size = 0.4 * nobs * nvars)] <- 0
x <- matrix(xvec, nrow = nobs)
beta <- rnorm(nvars / 3)
fx <- x[, seq(nvars / 3)] %*% beta / 3
ty <- rexp(nobs, exp(fx))
tcens <- rbinom(n = nobs, prob = 0.3, size = 1)
y <- survival::Surv(ty, tcens)
fit1 <- glmnet(x, y, family = "cox")

# survfit object for Cox model where lambda = 0.1
sf1 <- survival::survfit(fit1, s = 0.1, x = x, y = y)
plot(sf1)

# example with new data
sf2 <- survival::survfit(fit1, s = 0.1, x = x, y = y, newx = x[1:3, ])
plot(sf2)

# example with strata
```

```
y2 <- stratifySurv(y, rep(1:2, length.out = nobs))
fit2 <- glmnet(x, y2, family = "cox")
sf3 <- survival::survfit(fit2, s = 0.1, x = x, y = y2)
sf4 <- survival::survfit(fit2, s = 0.1, x = x, y = y2,
               newx = x[1:3, ], newstrata = c(1, 1, 1))
```

`survfit.cv.glmnet` *Compute a survival curve from a cv.glmnet object*

Description

Computes the predicted survivor function for a Cox proportional hazards model with elastic net penalty from a cross-validated glmnet model.

Usage

```
## S3 method for class 'cv.glmnet'
survfit(formula, s = c("lambda.1se", "lambda.min"), ...)
```

Arguments

`formula`	A class `cv.glmnet` object. The object should have been fit with `family = "cox"`.
`s`	Value(s) of the penalty parameter lambda at which predictions are required. Default is the value s="lambda.1se" stored on the CV object. Alternatively s="lambda.min" can be used. If s is numeric, it is taken as the value(s) of lambda to be used.
`...`	Other arguments to be passed to `survfit.coxnet`.

Details

This function makes it easier to use the results of cross-validation to compute a survival curve.

Value

If `s` is a single value, an object of class "survfitcox" and "survfit" containing one or more survival curves. Otherwise, a list of such objects, one element for each value in `s`. Methods defined for survfit objects are print, summary and plot.

Examples

```
set.seed(2)
nobs <- 100; nvars <- 15
xvec <- rnorm(nobs * nvars)
x <- matrix(xvec, nrow = nobs)
beta <- rnorm(nvars / 3)
fx <- x[, seq(nvars / 3)] %*% beta / 3
```

```
ty <- rexp(nobs, exp(fx))
tcens <- rbinom(n = nobs, prob = 0.3, size = 1)
y <- survival::Surv(ty, tcens)
cvfit <- cv.glmnet(x, y, family = "cox")
# default: s = "lambda.1se"
survival::survfit(cvfit, x = x, y = y)

# s = "lambda.min"
survival::survfit(cvfit, s = "lambda.min", x = x, y = y)
```

`weighted_mean_sd` *Helper function to compute weighted mean and standard deviation*

Description

Helper function to compute weighted mean and standard deviation. Deals gracefully whether x is sparse matrix or not.

Usage

```
weighted_mean_sd(x, weights = rep(1, nrow(x)))
```

Arguments

`x`	Observation matrix.
`weights`	Optional weight vector.

Value

A list with components.

`mean`	vector of weighted means of columns of x
`sd`	vector of weighted standard deviations of columns of x

Index

∗ **Cox**
  Cindex, 9
  coxgrad, 13
  coxnet.deviance, 14
∗ **classification**
  assess.glmnet, 4
∗ **cross-validation**
  Cindex, 9
∗ **datasets**
  beta_CVX, 7
∗ **data**
  BinomialExample, 8
  CoxExample, 12
  MultiGaussianExample, 47
  MultinomialExample, 48
  PoissonExample, 55
  QuickStartExample, 61
  SparseExample, 62
∗ **models**
  assess.glmnet, 4
  bigGlm, 7
  Cindex, 9
  coef.glmnet, 10
  cv.glmnet, 16
  deviance.glmnet, 21
  glmnet, 27
  glmnet-package, 3
  glmnet.control, 36
  glmnet.measures, 42
  makeX, 45
  na.replace, 49
  plot.cv.glmnet, 52
  plot.glmnet, 53
  predict.cv.glmnet, 56
  print.cv.glmnet, 59
  print.glmnet, 60
∗ **model**
  coxgrad, 13
  coxnet.deviance, 14
∗ **package**
  glmnet-package, 3
∗ **regression**
  bigGlm, 7
  coef.glmnet, 10
  cv.glmnet, 16
  deviance.glmnet, 21
  glmnet, 27
  glmnet-package, 3
  glmnet.control, 36
  plot.cv.glmnet, 52
  plot.glmnet, 53
  predict.cv.glmnet, 56
  print.cv.glmnet, 59
  print.glmnet, 60

assess.glmnet, 4

beta_CVX, 7
bigGlm, 7
BinomialExample, 8

Cindex, 9
coef.cv.glmnet *(*predict.cv.glmnet*)*, 56
coef.cv.relaxed *(*predict.cv.glmnet*)*, 56
coef.glmnet, 10
coef.relaxed *(*coef.glmnet*)*, 10
confusion.glmnet *(*assess.glmnet*)*, 4
CoxExample, 12
coxgrad, 13
coxnet.deviance, 14
cv.glmnet, 16

dev_function, 21
deviance.glmnet, 21

elnet.fit, 23

family, *27*, *40*, *44*
fid, 25

get_eta, 26
get_start, 26
glmnet, 27, *38, 39*, *44*
glmnet-package, 3
glmnet.control, *18*, *24*, *29–31*, 36, *41*, *44*
glmnet.fit, 39
glmnet.measures, 42
glmnet.path, 43

makeX, 45
MultiGaussianExample, 47
MultinomialExample, 48
mycoxph, 48
mycoxpred, 49

na.replace, 49

obj_function, 50

pen_function, 51
plot.cv.glmnet, 52
plot.cv.relaxed *(*plot.cv.glmnet*)*, 52
plot.glmnet, 53
plot.mrelnet *(*plot.glmnet*)*, 53
plot.multnet *(*plot.glmnet*)*, 53
plot.relaxed *(*plot.glmnet*)*, 53
PoissonExample, 55
predict.coxnet *(*coef.glmnet*)*, 10
predict.cv.glmnet, 56
predict.cv.relaxed *(*predict.cv.glmnet*)*, 56
predict.elnet *(*coef.glmnet*)*, 10
predict.fishnet *(*coef.glmnet*)*, 10
predict.glmnet *(*coef.glmnet*)*, 10
predict.glmnetfit, 57
predict.lognet *(*coef.glmnet*)*, 10
predict.mrelnet *(*coef.glmnet*)*, 10
predict.multnet *(*coef.glmnet*)*, 10
predict.relaxed *(*coef.glmnet*)*, 10
print.bigGlm *(*print.glmnet*)*, 60
print.cv.glmnet, 59
print.cv.relaxed *(*print.cv.glmnet*)*, 59
print.glmnet, 60
print.relaxed *(*print.glmnet*)*, 60

QuickStartExample, 61

relax.glmnet *(*glmnet*)*, 27
response.coxnet, 61
rmult, 62
roc.glmnet *(*assess.glmnet*)*, 4

SparseExample, 62
stratifySurv, 63
survfit.coxnet, 63
survfit.cv.glmnet, 65

weighted_mean_sd, 66

x *(*beta_CVX*)*, 7

y *(*beta_CVX*)*, 7